



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Sri Harsha Vathsavayi

**Applying Genetic Algorithms for Software Design and
Project Planning**



Julkaisu 1437 • Publication 1437

Tampere 2016

Sri Harsha Vathsavayi

Applying Genetic Algorithms for Software Design and Project Planning

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 2nd of December 2016, at 12 noon.

ISBN 978-952-15-3856-8 (printed)
ISBN 978-952-15-3878-0 (PDF)
ISSN 1459-2045

Applying Genetic Algorithms for
Software Design and Project Planning

Abstract

Today's software systems are growing in size and complexity. This means not only increased complexity in developing software systems, but also increase in the budget and completion time. This trend will lead to a situation where traditional manual software engineering practices are not sufficient to develop and evolve software systems in an economic and timely manner. Automated support can aid software engineers in reducing the time-to-market and improving the quality of the software. This thesis work explores the application of genetic algorithms for automated software architecture design and project planning.

Software architecture design and project planning are non-trivial and challenging tasks. This thesis applies genetic algorithms to introduce automation into these tasks. The proposed genetic algorithm exploits reusable solutions, such as design patterns, architecture styles and application specific solutions for transforming a given initial rudimentary model into detailed design. The architectures are evaluated using multiple quality attributes, such as modifiability, efficiency and complexity. The fitness function encompasses the knowledge required for evaluating the architectures according to multiple quality attributes. The output from the genetic algorithm is an architecture proposal optimized with respect to multiple quality attributes.

A genetic algorithm has also been devised for assigning work across teams located in distributed sites. The genetic algorithm takes information about the target system and the development organization as input and produces a set of work distribution and schedule plans optimized with respect to cost and duration objectives. The fitness function considers the differences in teams and barriers created by global dispersion into account in evaluating the work assignment. In addition, the genetic algorithm also takes solutions that ease or hamper distributed development into account in allocating the work. The genetic algorithm has been further extended with Pareto optimality to find a set of suitable work distribution proposals in a tradeoff between project cost and duration. In the experiments, an electronic home control system was developed by a set of different organizations structures. The results demonstrate that the proposed genetic algorithm can create reasonable work distribution proposals that conform to the general assumptions about the nature of cost and project completion time, i.e., cost of the project can be reduced at the expense of project completion time and vice-versa.

In addition, variations have been made to the genetic algorithm approach to software architecture design. To accelerate the genetic algorithm towards multi-objective solutions, a quality farms approach has been developed. The approach uses the idea of cross breeding, where different individuals that are good with respect to one quality objective are combined for producing software architecture proposals that are good in multiple objectives. Also, to explore the suitability of other methods for software architecture synthesis, a constraint satisfaction approach has been developed. The approach models the software architecture design problem as a constraint satisfaction and optimization problem and solves it using constraint satisfaction techniques. This approach can provide rationale about why certain decisions are chosen in the proposed architecture proposals.

Tool support for genetic algorithm-based architecture design and work planning approaches has been proposed. It facilitates an end user to give input, view and analyze

the results of the developed genetic algorithm based approaches. The tool also provides support for semi-automated architecture design, where a human architect can guide the genetic algorithm towards optimal solutions. An empirical study has also been performed. It suggests that the quality of the proposals produced through semi-automated architecture design is roughly at the level of senior software engineering students. Furthermore, the project manager can interact with the tool and perform what-if analysis for choosing the suitable work distribution for the project at hand.

Preface

This work was carried out during 2010-2014 as part of the Darwin research project in the Department of Pervasive Computing at Tampere University of Technology. The work has been funded by the Academy of Finland, Graduate School on Software and Systems Engineering (SoSE), Tampere University of Technology, Science Foundation of the City of Tampere and Nokia Foundation.

First of all, I would like to thank my first supervisor, Professor Kai Koskimies for his excellent guidance, support and encouragement during the thesis work. In addition, I wish to thank him for showing confidence in me and for his valuable feedback all over the years of this thesis work. I also would like to thank my second supervisor Professor Kari Systä for the valuable discussions and guidance during the thesis work. I especially want to thank him for investing his time and effort in completing the final parts of the research. His support was invaluable while finalizing the thesis.

I am grateful to my colleagues who made this work possible. Special thanks to Outi Sievi-Korte for her valuable feedback, contributions and support during the thesis. I would also like to thank Hadaytullah for his contribution to the tool support developed during this thesis work.

I would like to thank Professor Tomi Männistö (Department of Computer Science, University of Helsinki) and Professor Filomena Ferucci (Department of Computer Science, University of Salerno) for pre-examining my thesis. I am also thankful to Professor Tommi Mikkonen for reviewing my thesis.

Finally, I would like to thank my wife Prathyusha for supporting me while writing the thesis. Special thanks to my friends Chiru and Pab, who have always encouraged me while doing the research. I also would like to thank my parents and friends for their everlasting support and encouragement.

Tampere, 12.10.2016
Sriharsha Vathsavayi

Contents

Abstract	ii
Preface	iv
Contents	v
List of Figures	viii
List of Tables	x
Terms and Abbreviations	xi
List of Included Publications	xii
Author's Contribution to the Publications	xiii

PART I - FOUNDATION	1
1 Introduction	2
1.1 Motivation	2
1.2 Research Questions	4
1.3 Thesis Approach	5
1.4 Research Method	7
1.5 Thesis Contributions	9
1.6 Organization of the Thesis	11
2 Background.....	13
2.1 Genetic Algorithms	13
2.2 Pareto Optimality	17
2.3 Constraint Satisfaction Techniques	18
2.4 Software Architecture Design	20
2.5 Software Design Patterns and Architectural Styles	21
2.6 Software Project Planning	22
PART II – GENETIC ALGORITHMS FOR SOFTWARE DESIGN AND PROJECT PLANNING	24

3	Genetic Algorithms for Software Architecture Design	25
3.1	Software Architecture Design using Genetic Algorithms.....	25
3.1.1	Input	26
3.1.2	Encoding Initial Design into Chromosome	27
3.1.3	Mutations and Crossover.....	28
3.1.4	Fitness Function.....	29
3.1.5	Selection	29
3.2	Application to an Example System.....	30
3.3	Summary	35
4	Genetic Algorithms for Planning Global Software Development Projects	36
4.1	GSD Work Distribution Meta-model.....	36
4.2	Using Genetic Algorithm for Planning GSD projects	38
4.2.1	Encoding Initial Work Distribution Model into Chromosome.....	39
4.2.2	Mutations and Crossover.....	39
4.2.3	Fitness Function.....	42
4.2.4	Experiments and Evaluation.....	43
4.3	Multi-Objective Project Planning with Genetic Algorithms.....	47
4.3.1	Pareto Optimal front.....	47
4.3.2	Experiments and Evaluation.....	47
4.4	Summary	50
5	Algorithmic Aspects of Software Architecture Synthesis	52
5.1	Using Quality-farms in Genetic Architecture Synthesis.....	52
5.1.1	Method	53
5.1.2	Experiments and Evaluation.....	53
5.2	Using Constraint Satisfaction Techniques for Software Design	56
5.2.1	Constraint Satisfaction and Optimization Approach to Software Design...	58
5.2.2	Experiment and Evaluation	59
5.3	Summary	62
PART III – TOOL SUPPORT FOR SOFTWARE DESIGN AND PROJECT		
PLANNING		63
6	Tool Support	64
6.1	User Interface of Darwin.....	64
6.2	Architecture of Darwin	66
6.3	Use of Darwin.....	67

6.4	Summary	70
7	Interleaving Human Decision Maker and Automated Support.....	71
7.1	Semi-Automated Architecture Design Process	71
7.1.1	Tool Mechanisms for Interactive Architecture Design.....	72
7.1.2	Example.....	73
7.1.3	Empirical Study	74
7.2	Interactive Support for Work Distribution Decisions.....	75
7.3	Summary	77
PART IV	– CLOSURE.....	78
8	Related Work.....	79
8.1	Software Architecture Design	79
8.1.1	Studies using Meta-heuristics for Software Architecture Design.....	79
8.1.2	Studies using Constraint Satisfaction Techniques for Software Design.....	82
8.2	Population Initialization in Genetic Algorithm	83
8.3	Project Planning.....	84
9	Limitations.....	87
9.1	General Limitations	87
9.2	Software Architecture Synthesis	88
9.3	Work Distribution in GSD Projects	88
10	Conclusions	90
10.1	Thesis Questions Revisited	90
10.2	Future Work.....	92
10.3	Final Remarks.....	93
References		95
Appendices.....		108
Paper 1		117
Paper 2.....		118
Paper 3		119
Paper 4.....		120
Paper 5.....		121
Paper 6.....		122

List of Figures

Figure 1. A chromosome representation.....	14
Figure 2. Genetic algorithm flow chart	15
Figure 3. Mutation operation	15
Figure 4. Crossover operation	16
Figure 5. Example of Pareto optimal solutions (maximizing objectives)	18
Figure 6. Placing apples, oranges and mangoes in a cubicle	19
Figure 7. An abstract view on the software architecture design process [Jansen, 2008]	20
Figure 8. UML-based design process for applying genetic algorithm for software architecture design.....	26
Figure 9. Example of a chromosome with supergenes	27
Figure 10. List of fields in a supergene	27
Figure 11. Abstract use cases of ACVM	30
Figure 12. Sequence diagram for use case refill finished stock.....	31
Figure 13. A fragment of initial design for ACVM	32
Figure 14. Initial design of ACVM	33
Figure 15. Fitness graph of ACVM.....	34
Figure 16. Proposal for ACVM.....	35
Figure 17. Overview of GSD work distribution meta-model.....	36
Figure 18. Dependency between components	37
Figure 19. Overview of genetic algorithm approach for planning GSD projects	38
Figure 20. Structure of a supergene SG_i	39
Figure 21. Structure of a chromosome (collection of supergenes)	39
Figure 22. Overview of change team mutation	40
Figure 23. Overview of change development order mutation	41
Figure 24. Overview of crossover operation	41
Figure 25. Ehome system with effort, skills and precedence relationships.....	44
Figure 26. Team structure	44
Figure 27. Best work distribution and schedule plan when duration is optimized	45
Figure 28. Best work distribution and schedule plan when cost is emphasized	46
Figure 29. Estimated duration and cost for Test 1) duration is emphasized, and Test 2) cost is emphasized	46
Figure 30. Resources used for test 1 and test 2.....	48
Figure 31. Non-dominated solutions of test 1 and test 2.....	48
Figure 32. A work distribution result of test 2	49
Figure 33. Characteristics of New York and London teams, New York and Bangalore teams	50
Figure 34. Non-dominated solutions of test 1 and test 2.....	50
Figure 35. Farm approach for optimizing two quality objectives	53
Figure 36. Modifiability and Efficiency farms	54
Figure 37. Fitness graph of second stage genetic algorithm (i.e., farm approach)	55
Figure 38. Sub-fitness graphs of stage two (i.e., farm approach)	55
Figure 39. Fitness graph of normal genetic architecture synthesis	56
Figure 40. Overview of applying Adapter pattern to coffee machine subsystem of ehome.....	58
Figure 41. Proposal for ehome system.....	61
Figure 42. Darwin user interface.....	65
Figure 43. Darwin for planning projects.....	66

Figure 44. Darwin plugin.....	67
Figure 45. A fragment of proposed architecture for ehome	69
Figure 46. A fragment of proposed work distribution proposal for ehome	70
Figure 47. Incremental semi-automatic architecture generation.....	72
Figure 48. Darwin user interface with architecture view	73
Figure 49. Interactive architecture design process with Darwin.....	74
Figure 50. Multi-objective fitness graph view of Darwin	76
Figure 51. Work distribution proposal of solution P1	77
Figure 52. Work distribution proposal of solution P2.....	77

List of Tables

Table 1. Design science research guidelines (Hevner et al. 2004)	8
Table 2. Chapters and corresponding thesis contributions	12
Table 3. List of software engineers	14
Table 4. Applied mutations.....	28
Table 5. Mutations applied for work planning.....	40
Table 6. Modeling software architecture design as a constraint satisfaction and optimization problem.....	59
Table 7. Results of experts evaluation.....	75

Terms and Abbreviations

ACVM	Automatic Chocolate Vending Machine
AI	Artificial Intelligence
AQOSA	Automated Quality-driven Optimization of Software Architecture
ArchE	Architecture Environment
CASE	Computer Aided Software Engineering
COTS	Commercial off-the-shelf
CRA	Class Responsibility Assignment
CSOP	Constraint Satisfaction and Optimization Problem
CSP	Constraint Satisfaction Problem
GSD	Global Software Development
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
MPGA	Multi-Population Genetic Algorithm
MVC	Model View Controller
NSGA II	Non-dominated Sorting Genetic Algorithm II
PaLM	Propagation and Learning with Move
SPEA II	Strength Pareto Evolutionary Algorithm 2
SRF	Software Renovation Framework
UML	Unified Modeling Language
VEGA	Vector Evaluated Genetic Algorithm

List of Included Publications

- P1. H. Hadaytullah, S. Vathsavayi, O. Räihä, and K. Koskimies. Tool Support for Software Architecture Design with Genetic Algorithms. In *Proceedings of 5th International Conference on Software Engineering Advances (ICSEA' 10)*, 2010, IEEE Computer Society Press, pp. 359–366.
- P2. S. Vathsavayi, O. Räihä, and K. Koskimies. Using Quality Farms in Multi-Objective Genetic Software Architecture Synthesis. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC'12)*, 2012, IEEE Press, pp. 2130–2137.
- P3. S. Vathsavayi, H. Hadaytullah, and K. Koskimies. Interleaving human and search-based software architecture design. *Proceedings of the Estonian Academy of Sciences*, **62** (1), 2013, pp. 16-26. This publication is a revised version of the article with same name appeared in the *12th Symposium on Programming Languages and Software Tools (SPLST'11)*, 2011.
- P4. S. Vathsavayi, O. Sievi-Korte, K. Koskimies, and K. Systä. Using Constraint Satisfaction and Optimization for Pattern-Based Software Design. In *Proceedings of the 23rd Australasian Software Engineering Conference (ASWEC'14)*, 2014, IEEE Computer Society Press, pp. 29–37.
- P5. S. Vathsavayi, O. Sievi-Korte, K. Koskimies, and K. Systä. Planning Global Software Development Projects Using Genetic Algorithms. In *Proceedings of 5th Symposium of Search Based Software Engineering (SSBSE'13)*, 2013, Springer LNCS **8084**, pp. 269–274.
- P6. S. Vathsavayi, O. Sievi-Korte and K. Systä. Tool Support for Planning Global Software Development Projects. In *Proceedings of IEEE International Conference on Computer and Information Technology (CIT'14)*, 2014, IEEE Computer Society Press, pp. 458-465.

The permission of the copyright holders of the original publications to reprint them in this thesis is hereby acknowledged. P1 has already appeared in one of the co-author's Ph.D. thesis.

Author's Contribution to the Publications

The research work of this thesis was carried out at Department of Pervasive Computing, Tampere University of Technology. The author of this thesis has been member of a research group applying search-based techniques for solving software engineering problems. The other members of the group are Mrs. Outi Sievi-Korte (Ph.D) and Mr. Hadaytullah (Dr.Tech). The research group has been in close collaboration under the guidance of thesis supervisors Prof. Kai Koskimies and Prof. Kari Systä. However, the author's contribution to all of the publications has been essential and is discussed in detail in the following paragraphs.

Publication P1 introduces software architecture design using genetic algorithms and presents the Darwin tool environment for genetic software architecture design. The author of this thesis is one of the main contributors in accomplishing a tool for genetic synthesis of software architectures. The author and Hadaytullah have implemented the tool. The author is responsible for conducting experiments using the tool. Moreover, the author is also responsible for developing an approach that uses UML diagrams for expressing the input required for genetic software architecture design. The genetic algorithm was integrated into the tool by Outi Sievi-Korte and the author of this thesis has made modifications to the tool to be in line with the genetic algorithm. Professor Kai Koskimies acted as supervisor and was involved in designing the experiments.

Publication P2 presents the quality farm approach for optimizing the genetic software architecture design with respect to multiple objectives. The idea to employ quality farms for accelerating the progress of genetic software architecture design in finding optimal solutions was originated by the author. The author of this thesis is the main author of this publication and is responsible for implementation and experiments. Other authors acted as supervisors and were involved in designing the experiments.

Publication P3 describes an interactive software architecture design process, where a human architect can interact with the genetic software architecture design process. This publication also describes an empirical study where interactively produced architecture proposals are compared with architecture proposals made by senior software engineering students. The author is the main contributor to the interactive architecture design process. The contributions include designing a new mechanism to take existing architecture as input and implementation of tool extensions. The author of this thesis also conducted experiments and performed the evaluation of the approach. Other authors acted as supervisors and were involved in designing the experiments.

Publication P4 introduces the application of constraint satisfaction techniques to software architecture design. The idea to employ constraint satisfaction and optimization for software architecture design was originated by the author. The contributions include design and implementation of constraint satisfaction and optimization approach to software architecture design. Moreover, the experiment design and evaluation of the approach was carried out by the author. Other authors acted as supervisors and were involved in designing the experiments.

Publication P5 describes the application of genetic algorithms for planning GSD projects. The author is the main contributor in applying genetic algorithms for planning GSD projects. The author and Outi Sievi-Korte have contributed to the design of the

genetic algorithm. The author contributions include design and implementation of mutations and fitness functions. Other authors acted as supervisors and were involved in designing the experiments.

Publication P6 presents the tool support for planning GSD projects. The author is responsible for extending the tool to support project planning in GSD context. The author is also responsible for developing a meta-model, which describes the main concepts involved in allocating work to a GSD project. Moreover, the author has applied Pareto optimality to GSD project planning problem. The result analysis and evaluation of the approach was carried out by the author. Other authors acted as supervisors and were involved in designing the experiments.

PART I - FOUNDATION

This part describes the motivation for this study and introduces the research questions, research method and main contributions. It also gives the essential background information for understanding of this thesis work.

1 INTRODUCTION

1.1 Motivation

Software systems are getting bigger and more complex with the passage of time. At the same time, there is a growing need for the production of software that meets requirements, such as performance, reliability, maintainability and cost [Roy and Veraart, 1996]. As a consequence of the increasing complexity and the need for quality software products, more and more efforts need to be spent to develop software. This has made software development an expensive activity and also has increased the time-to-market, which is undesirable given the financial constraints and fierce competition in the software industry. The main factor contributing to high development costs and longer development cycle is the vast amount of human effort required for producing the software. Furthermore, humans are limited by the Golden Hammer syndrome [Brown et al., 1998], a person tends towards applying solutions that he or she is familiar with or has successfully applied in the past. It is quite possible that in choosing a solution to the problem the software architects, project managers, software developers and testers may not consider all the potential solutions to the problem. This may reduce the quality of the human-made solutions in developing the software.

In order to reduce the cost to develop the software and shorten the time-to-market, researchers are searching means for automation. Many studies have focused on developing techniques that can fully or partially automate various activities in software engineering [Kramer and Magee, 1985; Selonen et al., 2001; Garlan et al., 2004; Harman et al., 2012, Aleti et al., 2013]. Automation improves the productivity of software engineers by stopping them from doing the redundant and repetitive work. Moreover, machines can objectively select a solution without having bias to a specific solution. Although it is a challenging task to develop automated procedures for all activities of software development life cycle, employing automated procedures for conducting some software engineering activities is still worth achieving.

Automation has been employed in different activities of software engineering life cycle, such as requirement analysis, planning, design, testing and maintenance (e.g. [Garlan et al., 2004], [Barreto et al., 2008], [Harman et al., 2012] and [Keeffe and Cinneide, 2006]). Various computer aided software engineering (CASE) tools and integrated development environments (IDE's) are used for modeling the design and for code generation and testing. For planning the projects different project management tools are used (e.g. MS Project [Microsoft project, 2015], OpenProject [OpenProject, 2015] and Jira [Jira, 2015]). However, there is little automation support developed for early software engineering activities, such as design and project planning. These activities are critical for fulfilling the quality requirements of software and for reducing the time-to-market. Automation support is especially beneficial for these activities, as finding an optimal solution for these activities is a challenging task and often beyond human

capabilities [Aleti et al., 2013]. The goal of this thesis is to construct automated procedures for software architecture design and project planning activities.

Software architecture design is one of the most crucial and challenging activities in the development of software systems. Software architectures are created and maintained in complex environments. While designing the architecture, software architects make multiple architecture decisions, which correspond to the evaluation of a number of potential conflicting architectural alternatives that can be employed. These architecture decisions eventually make up the architecture of a system. This resulted in viewing software architecture design as a result of architecture design decisions made over time [Jansen and Bosch, 2005]. These decisions may be related to which *design patterns* [Gamma et al., 1995] and *architecture styles* [Shaw and Garlan, 1996] can be applied in the context of system and choosing the appropriate commercial off-the-shelf (COTS) components to meet system requirements. Moreover, the decisions can also be related to the application domain of the system and other infrastructure selections needed to satisfy system requirements [Jansen, 2008]. However, with the increasing system complexity, the space of design options the software architects have to consider for an optimal architecture design is growing continuously [Aleti et al., 2013]. This leads to a large design space that is difficult to explore manually and makes software architecture design a challenging task. Thus, an approach that can automatically explore the design space and apply suitable architecture decisions can be valuable to the architect in software architecture design process. In addition to the productivity of the architect, it can also improve the quality of the resulted architecture. This is based on the fact that the automated approaches may have access to and consider without prejudice a much larger solution and knowledge base than a human decision maker, who often suffers from the Golden Hammer syndrome [Brown et al., 1998].

Prior to starting a software project the management needs to decide and plan how to develop the software. This involves identifying the resources needed for developing the project, distribution of work to available teams, deciding when the teams carry out the work and how to monitor the progress of work [Chang et al., 2001]. The large number of market constraints and different dimensions involved in software development makes project planning a complex task. However, to be successful in a competitive world, the organizations have to effectively plan the projects.

In order to reduce the software development costs and to provide more functionality and higher quality software faster, software organizations have moved towards developing software with teams located in different geographical regions. This kind of software development is known as global software development (GSD). When compared to collocated software development, the teams in a GSD project are often located in different geographical locations, come from different cultural backgrounds and work on the project in different time-zones. GSD has become a common practice in the software development industry [Smite et al., 2010]. However, despite its widespread use GSD still imposes many challenges. The physical distance affects the communication between the teams, but other conditions complicate the situation even more. The language barriers and different working styles can cause delays and affect working relationships. Moreover, when collaborating in different time zones, controlling and coordinating of the development activities at all sites are challenging. Also, the collaborating organizations may not share the same objectives, leading to misunderstandings and miscommunication between the teams [Lamersdorf, 2011]. Project management has a central role in coordinating and controlling how software is developed. Effective project

management can alleviate many of these challenges. Especially, work allocation to distributed teams is crucial, as it establishes the communication structure between the teams, and has direct impact on the communication and coordination problems.

Work allocation is challenging in GSD, as teams are located in different geographical regions with different time-zones and cultural distances among them. Moreover, the software architecture of the system should be taken into account in allocating the work. In fact both the phases are inter-related. The software architecture of the system determines the potential work units and their dependencies. These dependencies between the work units impose the communication and coordination effort required for realizing the architecture. This information has to be taken into account in allocating work to teams; otherwise work allocation may result in poor dissemination of work to teams and can increase the communication overhead between the teams. To minimize the communication and coordination effort required between the teams, software architecture is often designed in such a way that it mimics the structure of the organization, which is also termed as Conway's law [Conway, 1968].

It is difficult to manually find an optimal work distribution plan for GSD projects, as the number of possible work distribution plans is huge and each plan affects the project outcome differently. The problem is intertwined with the problem of architecture design, which further complicates the problem. In particular, the GSD practitioners need automation support for studying how different factors, for example, team skills, team performance and motivation influence the chosen work distributions and project outcome. Furthermore, the automation support can not only aid the project manager in searching for the optimal solutions, but can also propose solutions which a project manager has overlooked.

The automated support for software architecture design and project planning can aid software architects and project managers in coming up with optimal architecture proposals and project plans and can shorten the time-to-market. However, the process of automatically generating the architecture proposal of a system from requirements is still unclear. In this thesis I will describe such an approach that can aid software architect in designing the system. Similarly, work allocation in GSD is often based on the cost considerations and the other factors (e.g. team characteristics, cultural differences and time-zones) are not considered. An approach considering multiple aspects of the GSD characteristics into work distribution and supporting the project manager to study different "what-if" scenarios during the phase of project planning is still missing. In this thesis I will present such an approach that can support the project manager in creating effective project plans and in studying different project planning scenarios.

1.2 Research Questions

The central objective of this thesis is to develop automated procedures for software architecture design and project planning activities. The automated approaches can aid the architects and project managers in developing high quality software architecture designs and in creating effective project plans.

With the increasing system scale, the number of architecture solutions and quality attributes that need to be considered are growing continuously [Aleti et al., 2013]. Similarly, a variety of different possible work distribution plans and their influence on project outcome need to be evaluated in order to choose a best work distribution plan

that suits the needs of the project at hand. In case of large software projects, the search space of possible architecture designs and work plans is far too large to search manually or adopt an exhaustive search method for finding an optimal architecture proposal and work distribution plan. Due to huge search space, often it is not feasible to perform an exhaustive search in polynomial time. On the other hand, search-based techniques, for example, genetic algorithms, hill climbing and simulated annealing are proved to be effective for solving problems with huge search space [Husbands, 1998]. In particular, genetic algorithms are global search algorithms, sampling many points in the search space at once and offer more robustness in finding solutions compared with other local search techniques, such as hill climbing and simulated annealing, which operate with reference to one solution at any time [Harman et al., 2012]. They have already been employed for automatically finding solutions to many software engineering problems (e.g. Jones et al., 1998, Wegener et al., 1997, Harman and Jones, 2001, Amoui et al., 2006) and are applied in solving software design problems as well [Räihä, 2010; Aleti et al., 2013].

This thesis explores using genetic algorithms for developing the automated support for software architecture design and project planning activities. We have identified different sub-goals that we need to aim for in order to fulfill our final goal. They are mostly concerned about applying genetic algorithms for software architecture design and planning GSD projects as well as enabling human decision maker to exploit automation support in designing and planning projects. Moreover, the developed approaches should provide support for optimizing multiple objectives. From these goals several sub-questions are formulated:

RQ-1: How to apply genetic algorithms for developing automated support for software architecture design and for planning GSD projects?

RQ-2: How can an end user (with little knowledge of search-based techniques) express the input required to apply genetic algorithms for software architecture design and planning projects? How to present the results to an end user?

RQ-3: How can a human decision maker and genetic algorithm-based automated support collaborate in solving software architecture design and project planning problems?

RQ-4: How can genetic algorithm-based approaches be made more effective for the multi-objective software architecture design and project planning?

1.3 Thesis Approach

To answer RQ-1, this work views software architecture design as a set of architecture design decisions [Jansen and Bosch, 2005]. In many cases, good software architectures are obtained by applying decisions related to design patterns [Gamma et al., 1995], architecture styles [Shaw and Garlan, 1996] and reference architectures used in the context of a particular system, rather than making completely new decisions [Avgeriou et al., 2007]. In this perspective, the software architecture design can be seen as a search problem of finding architecture decisions that improve the quality of the architecture in the context of a particular system. This thesis applies *genetic architecture synthesis* [Räihä et al., 2008] for designing the architecture of a system at hand [P1, P2, P3]. The approach takes the initial functional decomposition and operation characteristics of the

system as input and then transforms this input by applying different design patterns [Gamma et al., 1994] and architecture styles [Shaw and Garlan, 1996] to satisfy the quality requirements of the system.

The heuristic search techniques have a drawback that they lack good explanation for the generated solution, which would be helpful in some cases. For example, the explanation behind a proposed architecture solution enables the decision maker to reapply that solution in similar kind of situations and is also very useful for reuse of design experience. To develop such an approach, this thesis explored the application of constraint satisfaction methods to software architecture design in publication [P4]. The proposed approach takes the initial functional decomposition and operation characteristics of the system as input and associates a set of solutions, i.e., design patterns and architecture styles to it. The problem of finding suitable architecture decisions for a target system is modeled as a constraint satisfaction and optimization problem. An attractive aspect of the approach is that it provides rationale for the chosen architecture decisions.

In order to succeed in a highly competitive software industry, software companies need to create efficient project plans. An important aspect of project planning is allocating work to available resources in the organization. This becomes even more challenging in the context of GSD [Lamersdorf, 2012], where teams are located in multiple physical locations and work in different time zones. Thus, to aid a project manager in planning GSD projects, this thesis applied genetic algorithms for automatically planning GSD projects [P5]. The approach takes information about distributed team characteristics and available work packages as input and applies the genetic algorithm to discover optimal work distributions and schedule to develop the software, together with solutions that ease the communication between teams. This approach is expected to answer RQ-1.

Tool support is essential for performing multiple experiments using the genetic algorithm-based approaches and to examine the results in detail. Also, the end users, i.e., software developers should be able to apply and understand the results produced by the developed genetic algorithm-based approaches without requiring much knowledge about the underlying techniques. For this purpose, tool support *Darwin* has been developed [P1, P6]. The Darwin tool provides user interface for the end user to express the input required for applying genetic algorithms to software architecture design and project planning. Moreover, it provides controls to monitor the algorithm's progress and deeply study the obtained results. The Darwin tool support is expected to answer RQ-2.

Two collaboration mechanisms have been proposed to answer RQ-3. The first mechanism enables an architect to introduce her feedback into the Darwin tool [P3]. The architect can judge the quality of the architecture proposals generated by the Darwin tool and can propose possible changes to be considered while reapplying the genetic algorithm. These possible changes are taken into account by the genetic algorithm for producing more fine-grained architectures. The second mechanism provides the project manager guidance in choosing the suitable work distribution plan for a project at hand [P6]. Instead of presenting a single solution to the problem, the mechanism provides the project manager with a set of optimal solutions giving information about how different solutions influence the project objectives. These solutions can be then closely examined by the project manager in deciding on the final work distribution plan to be adopted.

The software architecture design and project planning problems are multi-objective problems, where objectives that have to be optimized are often competing with each other. To cope with the multi-objective nature of software architecture design and project planning, this thesis has studied two methods. The first method introduces a quality farms approach to accelerate the genetic architecture synthesis for multiple quality objectives [P2]. The approach uses the idea of cross breeding for producing software architecture proposals that are good in multiple objectives. The second method applies *Pareto optimality* [Deb, 1999] to find a set of work distribution proposals in a tradeoff between multiple objectives [P6]. These methods are expected to answer RQ-4.

1.4 Research Method

The main focus of this thesis is to develop automated procedures for software architecture design and project planning activities. Design science is a research paradigm that is especially aimed at creating new artifacts. In the research presented in this thesis, design science paradigm is followed in developing the artifacts for automating software architecture design and project planning activities.

Design science aims at creation of useful artifacts for understanding and addressing general problems in a business setting [Hevner et al., 2004]. The created artifacts include: constructs, methods, models and instantiations. The application of these artifacts will create new innovations or solutions to existing problems. Design science aims at developing solutions to new problem contexts and evaluating the benefits and drawbacks of the solutions in this new context, rather than explaining the existing problems. This thesis follows Hevner et al.'s [2004] guidelines for conducting design science research. These guidelines are presented in Table 1.

The first guideline specifies that design science research must produce viable artifacts. This guideline is covered in this thesis by constructing the following artifacts:

1. A method for expressing the input for genetic architecture synthesis using UML diagrams. It uses use case, sequence and class diagrams for providing the input needed to apply genetic algorithms for architecture synthesis. It is evaluated by conducting experiments (Chapter 3) and is shared with the research community in publication [P1].
2. A method for planning GSD projects using genetic algorithms. It takes team characteristics and available work packages as input and applies genetic algorithms for finding the optimal work distribution plans and schedule plans with respect to the project's cost and duration goals. It is evaluated by conducting multiple experiments (Section 4.2) and is presented to the research community in publications [P5] and [P6].
3. A method to automatically produce the software architecture design of a system. The problem of finding suitable architecture decisions with respect to the quality requirements of the system is modeled as a constraint satisfaction and optimization problem. The method is evaluated by designing the software architecture of an example system (Section 5.2) and is discussed in publication [P4].

4. A tool named Darwin was developed for applying genetic algorithms for architecture synthesis and project planning. The usage of the tool is demonstrated by applying to an example system (Chapter 6) and is discussed in publications [P1] and [P6].
5. Two different mechanisms to allow decision maker to interact with the genetic algorithm-based automated approaches. These methods enable the decision maker to analyze the automatically produced results and guide the search process of genetic algorithms. They are evaluated by conducting an empirical study (Section 7.1) and are presented to the research community in publications [P3] and [P6].
6. A method for multi-objective project planning using genetic algorithms. It applies Pareto optimal genetic algorithm for finding work distribution and schedule plans that satisfy multiple objectives. The method is evaluated by studying different project scenarios (Section 4.3) and is shared with the research community in publication [P6].
7. A method for accelerating genetic algorithms to find multi-objective solutions. It is evaluated by applying to genetic algorithm based architecture design (Section 5.1) and is discussed in publication [P2].

Table 1. Design science research guidelines (Hevner et al. 2004)

<i>Guideline</i>	<i>Description</i>
Guideline 1: Design as an artifact	Design-science research must produce a viable artifact in the form of a construct, a method, a model, or an instantiation.
Guideline 2: Problem relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

The second guideline suggests that the design science research must develop technology based solutions to important and relevant business problems. This thesis follows this guideline by targeting relevant business problems. The problems targeted in this thesis are software architecture design and project planning in GSD. Software architecture design is very crucial for the success of the software development and plays a main role in evolving the software for future business needs. GSD has become a common practice for software organizations. However, despite its widespread use, GSD still poses many communication and coordination challenges. Organizations require efficient project planning practices to overcome these challenges and execute GSD projects successfully.

According to the third guideline, the artifacts must be rigorously demonstrated using well-executed evaluation methods. This thesis covers this guideline by conducting multiple controlled experiments [Zelkowitz and Wallace, 1997] under laboratory settings using the developed artifacts and discussing the limitations of the developed artifacts (see Chapter 9). The genetic algorithm approach for planning GSD projects is evaluated using project planning scenarios, as reported in [P5] and [P6]. The constraint satisfaction and optimization approach to software architecture design is evaluated by designing the architecture of an example system, as presented in [P4]. The tool support is evaluated by applying it to design the architecture of an example system and plan an imaginary GSD project, as reported in [P1, P3 and P6]. The techniques to optimize multiple objectives are evaluated by applying them to an example system, as presented in [P2] and [P6].

The fourth guideline suggests that the design research must provide clear and practical contributions. This thesis follows this guideline by discussing the possible practical and scientific contributions of the developed artifacts, together with their limitations and weaknesses. The fifth guideline research rigor suggests that artifacts should be developed based on established knowledge foundations. This thesis utilizes this guideline by using established algorithms, such as genetic algorithms and constraint satisfaction techniques for developing the artifacts. Moreover, this thesis uses well established solutions, such as design patterns [Gamma et al., 1995] and architecture styles [Shaw and Garlan, 1996] in developing the artifacts. The sixth guideline searching for an efficient artifact is satisfied by searching for the appropriate technique and appropriate design patterns [Gamma et al., 1995] and architecture styles [Shaw and Garlan, 1996] for developing the artifacts. The seventh guideline is fulfilled by sharing the research results with the scientific community in international conferences and in publications.

1.5 Thesis Contributions

In the thesis, several research contributions were made. The contributions of this thesis are described in the following.

1. An approach to use a set of unified modeling language (UML) diagrams for expressing the input for genetic software architecture synthesis.

This contribution describes a method to use UML diagrams, such as use case, sequence and class diagrams for expressing the input required for genetic architecture synthesis approach. The genetic algorithm then transforms this input into an architecture proposal by applying suitable different design patterns [Gamma et al., 1994] and architecture styles [Shaw and Garlan, 1996] to satisfy the quality requirements of the

system. Using well-known modeling language for expressing the input makes it easy for the software developers to use the genetic architecture synthesis approach. Moreover, it opens the possibility to integrate the genetic architecture synthesis approach with existing CASE tools. This contribution is presented in publication [P1].

2. A method for automatically designing the software architecture of a system using constraint satisfaction and optimization.

This contribution demonstrates how to apply constraint satisfaction and optimization technique to automatically design the software architecture of a system. The approach has the capability to reason about the chosen design decisions. This support is beneficial for understanding the rationale behind the chosen design decisions and can support reusing design knowledge in similar situations. Moreover, this kind of support is helpful for managing architectural knowledge in the organizations and is valuable in long run. This contribution is presented in publication [P4].

3. A method for planning GSD projects using genetic algorithms.

The thesis presents a concrete approach based on genetic algorithms for finding a set of optimal work assignment plans and schedule plans for GSD projects. Moreover, the approach can also propose architecture solutions that ease the communication between inter-site teams. This is one of the first approaches to apply genetic algorithms for studying the GSD project planning problem. This work expands the research on search-based techniques for software engineering to the field of GSD, which has not been properly explored, by former research. This contribution is presented in publications [P5] and [P6].

4. A tool for software design and project planning.

A tool support for enabling software developers to use the genetic algorithm-based software architecture design and project planning approaches is discussed in this thesis. When compared to existing tools for automated software architecture design, the benefits of Darwin tool is a simpler starting point, i.e., a UML based functional decomposition of the system is given as input and is enough for generating an architecture proposal. Furthermore, the tool can be used without requiring much knowledge related to search algorithms. This contribution is presented in publications [P1] and [P6].

5. Methods for collaboration of human decision maker and automated support.

This thesis suggests a set of tool mechanisms to allow a human decision maker interact with the automated support in solving the architecture design and project planning problems. These mechanisms make it possible to leverage human decision maker knowledge in the automatic software architecture generation process. This contribution is presented in publications [P3] and [P6].

6. A technique for multi-objective genetic architecture synthesis using quality farms.

This contribution suggests an approach to optimize and accelerate genetic architecture synthesis towards architecture proposals that satisfy multiple quality objectives. This approach introduces a new population initialization method to the genetic algorithm and can be easily applied to problem domains, where genetic algorithms are used for optimizing multi-objective problems. This contribution is presented in publication [P2].

7. A method for multi-objective project planning using genetic algorithms.

A Pareto optimal genetic algorithm for evaluating GSD work distribution plans with two objectives, cost and duration is presented in this thesis. When compared to existing approaches for planning GSD projects, this method provides the possibility for the project manager to study different “what-if” scenarios in the context of GSD. This kind of support can give insightful knowledge for decision making on suitable solution to be available project constraints. This contribution is presented in publication [P6].

In addition to the publications included in the thesis, the topics of this thesis are discussed by the author also in publication [Vathsavayi and Systä, 2016].

1.6 Organization of the Thesis

This thesis includes an introduction, six original articles published previously and an appendix. The introduction is divided into four parts. The first part includes two chapters. After this introductory chapter, the background information fundamental to this thesis is presented in Chapter 2.

The second part is dedicated to application of genetic algorithms to software design and project planning. The application of genetic algorithms to software architecture design is presented in Chapter 3. Chapter 4 discusses the application of genetic algorithms for planning GSD projects. The investigations on the algorithm aspects of software architecture synthesis are presented in Chapter 5.

The third part of this thesis focuses on the tool support for using the developed approaches. Chapter 6 discusses the developed tool support Darwin. The techniques for collaborating human decision maker and automated support are presented in Chapter 7.

The fourth part concludes this thesis. The previous studies in this area are presented in Chapter 8. Chapter 9 discusses the limitations of the study. The main conclusions of the thesis are summarized in Chapter 10.

The chapters and the contributions they address and the associated publications and produced artifacts are presented in Table 2. For example, the first row in the table can be

read as Chapter 3 describes the first contribution of the thesis and is discussed in the publication [P1] and corresponds to the artifact 1.

Table 2. Chapters and corresponding thesis contributions

		Contributions						
		CONTRIB 1	CONTRIB 2	CONTRIB 3	CONTRIB 4	CONTRIB 5	CONTRIB 6	CONTRIB 7
Chapters	3. GENETIC ALGORITHMS FOR SOFTWARE ARCHITECTURE DESIGN	[P1] Artifact 1						
	4. GENETIC ALGORITHMS FOR PLANNING GLOBAL SOFTWARE DEVELOPMENT PROJECTS			[P5, P6] Artifact 2				[P6] Artifact 6
	5. ALGORITHMIC ASPECTS OF SOFTWARE ARCHITECTURE SYNTHESIS		[P4] Artifact 3				[P2] Artifact 7	
	6. TOOL SUPPORT				[P1, P6] Artifact 4			
	7. INTERLEAVING HUMAN DECISION MAKER AND AUTOMATED SUPPORT					[P3,P6] Artifact 5		

2 BACKGROUND

Main background concepts are introduced in this chapter. It first gives a brief introduction to genetic algorithms and Pareto optimality, followed by summary of constraint satisfaction techniques. Next, the background of software architecture design and reusable solutions used in software architecture domain are introduced. Finally, the summary of software project management is presented.

2.1 Genetic Algorithms

Genetic algorithms were invented by John Holland [Holland, 1975]. Genetic algorithms are inspired by the ideas of natural selection and genetics. Genetic algorithms belong to the family of meta-heuristic search algorithms and are applied for solving problems with a very large search space. Genetic algorithms provide a sophisticated way to quickly find a good solution for a problem that is not feasible with deterministic search methods [Mitchell, 1996].

Some biological terms need to be known for understanding the genetic algorithms. All living organisms consist of cells, where each cell contains a set of *chromosomes*. A chromosome consists of number of individual structures called *genes*. A gene represents a particular property of an organism and the location, or *locus*, of the gene in chromosome structure determines which characteristic the gene represents. A gene may encode one or several different values of the particular characteristic it represents. The different values of genes are called *alleles*. Gathering all the chromosomes specifies the entire individual.

Genetic algorithms are a way of using the ideas of evolution in computer science. They operate with a set of individuals (or chromosomes). In the genetic algorithm, each individual contains only one chromosome. A set of individuals at a certain time point is called a *population*. Chromosomes are evolved through successive iterations called *generations*. During each generation, new chromosomes are formed by either merging two chromosomes using *crossover* operator or by modifying a chromosome using *mutation* operator. In order to know which individuals are better fit than others the *fitness* of the individual is calculated. The fitness indicates the probability that the organism will live to participate in the reproduction. The individuals for new generation are formed by *selection*, according to the fitness values. The algorithm ends when a *terminating condition* has been reached. The commonly used terminating conditions are fixed number of generations or certain budget of fitness evaluations.

Genetic algorithms are inspired by Darwinian evolution, in keeping with this analogy, a solution to the problem has to be first represented in terms of a chromosome. Binary encodings are the most commonly used encodings [Mitchell, 1996]. In binary encodings, a chromosome is represented using a bit vector, i.e., strings of ones and zeroes, where

every bit represents a gene of the chromosome. The other common way to encode a chromosome is to use string of natural numbers. The encoding of a chromosome can be done in different ways, for example, numeric and non-numeric [Michalewicz, 1992]. It is up to the genetic algorithm designer to choose the suitable encoding needed for the problem, such that the encoding can represent the solution to the problem.

The binary encoding of a chromosome can be illustrated using an example of creating a software team for carrying out a software project. Say, we have seven software engineers (as shown in table 3), each with an experience of e_i and a salary of s_i . The objective is to find an optimal team with respect to experience with maximum cost of 14000\$.

Table 3. List of software engineers

<i>E</i>	<i>S</i>
2	3100
1	2500
3	3500
4	3500
2	2500
5	4000
3	3000

For this problem, the chromosome can be represented using a vector of 7 bits, where 1 represents picking the software engineer represented by that gene, and 0 represents not picking the software engineer. Each index of the vector corresponds to a software engineer in the order they are listed in Table 3. For example, index 1 corresponds to software engineer with 2 years of experience and salary of 3100\$. The chromosome (shown in Figure 1) containing software engineers at loci 1, 3 and 4 can be one possible solution to the problem.

1	0	1	1	0	0	0
---	---	---	---	---	---	---

Figure 1. A chromosome representation

The work flow of a typical genetic algorithm is presented in Figure 2. The genetic algorithm usually starts with a random initial population. The population then goes through a set of mutation and crossover operations. The fitness function evaluates the quality of the individuals in each generation. Next, the selection operator is applied on the population, to select the individuals for the next generation. If the specified terminating condition is reached, the algorithm will be stopped. Otherwise, the new population will go through as many generations of crossovers, mutations and selections

as is needed until the terminating condition is reached. The best individual resulted after the terminating condition is considered as the best solution.

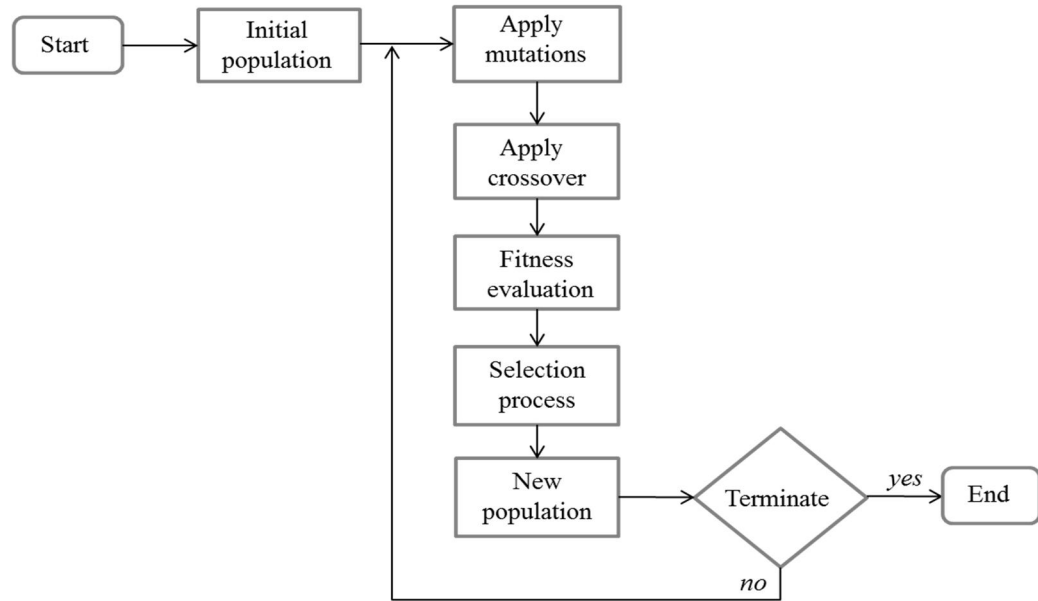


Figure 2. Genetic algorithm flow chart

A mutation creates new individuals by changing the alleles of an existing individual at a random locus. In a binary encoded chromosome, application of a mutation may flip a bit at randomly chosen locus, i.e., to change a bit from 1 to 0 or 0 to 1. For example, if chromosome A (shown in Figure 3) is mutated at the third locus, then the resultant chromosome looks like chromosome A'. Moreover, application of a mutation may also make a chromosome illegal. For example, if chromosome A is mutated at sixth locus, then the resulting chromosome will become illegal, as salary exceeds 14000\$. To ensure that the chromosome stays legal, different options can be used: checking whether a mutation is possible before performing it, constructing a corrective operator that will modify the result of a mutation, discarding the illegal chromosome from the population, or punishing the illegal chromosome heavily such that it is not considered for the population of the next generation.

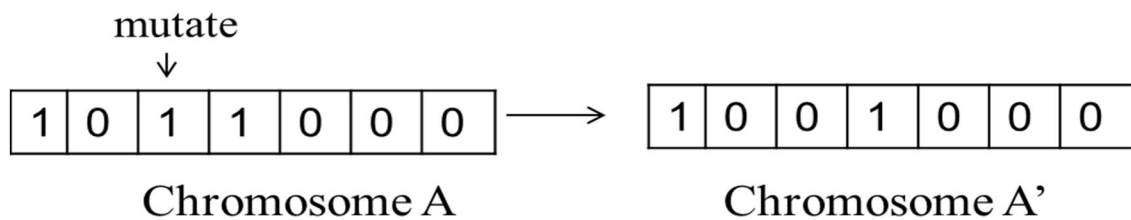


Figure 3. Mutation operation

The mutations are also given a probability, which specifies how likely the mutation will be applied to an individual. This is called the *mutation probability* or *mutation rate* [Mitchell, 1996]. If the mutation probability is 20%, then it means that approximately 20% of the individuals of a generation will be subjected to that mutation.

During a crossover operation, two offspring individuals are formed by exchanging the genes of two individuals. Similar to mutations, crossover is also applied to a randomly selected locus in a chromosome. One of the most commonly applied crossover operator is *single point crossover*, where new offspring is created by exchanging the subsequences before and after the selected locus [Mitchell, 1996; Michalewicz, 1992]. For example, two parent chromosomes for the team forming example discussed above can be as shown in Figure 4. If the third locus is selected as a crossover point, then the resultant offspring will be like child 1 and child 2 as shown in Figure 4. Crossover is also given a probability, i.e., a specified *crossover probability* or *crossover rate*. This probability specifies likelihood of an individual being subjected to crossover. Similar to mutations, a crossover may also make a chromosome illegal. In such situations, a corrective operator can be used to fix the chromosome.

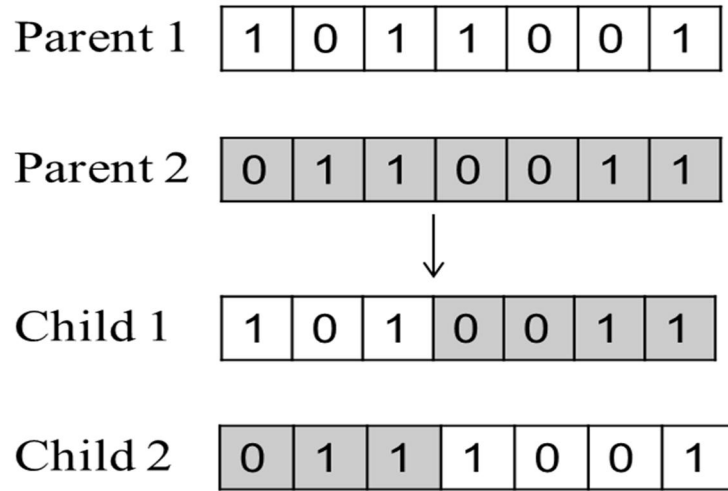


Figure 4. Crossover operation

Next, the fitness function for evaluating the quality of solution needs to be defined. The fitness function assigns each chromosome a value termed as “fitness”, which tells how well a chromosome solves the given problem. Thus, the fitness function is crucial for obtaining optimal solutions. If the problem at hand is concerned with the numerical data, then the fitness function can be detected from the problem itself. However, in the case of non-numerical data, the designer of the genetic algorithm must find other ways to evaluate non-numerical data [Mitchell, 1996]. For the team forming example, the fitness function is shown in equation (1).

$$f(x) = \sum_{i=1}^7 e_i \cdot x_i + \sum_{i=1}^7 s_i \cdot x_i \leq 14000 \quad (1)$$

where x_i is 1 if software engineer at locus i is selected and 0 otherwise. The terms e_i and s_i represent the experience and salaries of software engineers at locus i . As per the fitness function, the fitness for chromosome in Figure 1 would be 9.

Finally, a selection operator needs to be specified. As the genetic algorithm processes population of chromosomes, the population size always increases due to crossover, and a selection operator is needed for keeping consistency in the population size. For example, if 25 pairs of chromosomes are subjected to crossover in a population of size 100, then the resulting population size will become 150, as 50 new chromosomes created from

crossover (see Figure 4) are added to the population. The selection operator will determine the individuals who will survive to the next generation, and should thus be defined so that the individuals with better fitness are more likely to survive.

The simplest method to select individuals for next generation is to use *elitist selection*. This selects only very best individuals in terms of fitness. The elitist selection is easy to implement, one can simply discard the weakest individuals in the population. However, it may converge towards a local optimum. Another and more commonly used selection operator is fitness-proportionate selection, which can be implemented with a *roulette wheel* sampling [Mitchell, 1996; Michalewicz, 1992]. Each individual in the population is given a piece of slice in the wheel area, which is proportional to its fitness in the overall fitness of the population. This way, the individuals with higher fitness always have larger area, and thus have a higher probability of getting selected. The wheel is then spun for as many times as there are individuals needed for the population.

2.2 Pareto Optimality

Many software engineering problems are of multi-objective nature. The objectives that have to be optimized are often competing with each other. For example, in project planning, aiming early completion time with low cost causes conflict between objectives. In this situation, one simple way to optimize the objectives is using aggregate fitness function, as in the case of team forming example discussed earlier. An alternative approach to aggregated fitness function is to use Pareto optimality [Deb, 1999]. The Pareto optimal selection evaluates solutions separately for each objective, and instead of presenting a single optimal solution, a set of satisfactory solutions is presented.

Suppose that a problem to be solved has n fitness functions, f_1, \dots, f_n , where $f_i(x)$ evaluates a solution x for an individual objective. For example, let us assume that the goal is to maximize all the objectives. If the objectives to be optimized are conflicting with each other, then it is difficult to find a solution in the search space that is optimal with respect to all objectives. The Pareto optimality can be used to compare solutions in such situations. Under Pareto optimality, a solution x' can be said to be better than solution x , if

$$\forall i. f_i(x') \geq f_i(x) \quad \wedge \quad \exists i. f_i(x') > f_i(x) \quad (2)$$

In other words, solution x' is better than solution x if it is better according to at least one of the individual fitness functions and no worse according to all of the others.

Searching for solutions using Pareto optimality yields a set of solutions, where each member of the set is no worse than any of the others in the set, but also cannot be said to be better. These solutions are referred as *non-dominated* solutions. The set of non-dominated solutions among all the feasible solutions forms a *Pareto front*. The example of Pareto optimal solutions is shown in Figure 5. In the figure, solutions a, b, c and d are non-dominated solutions. These solutions form the Pareto front.

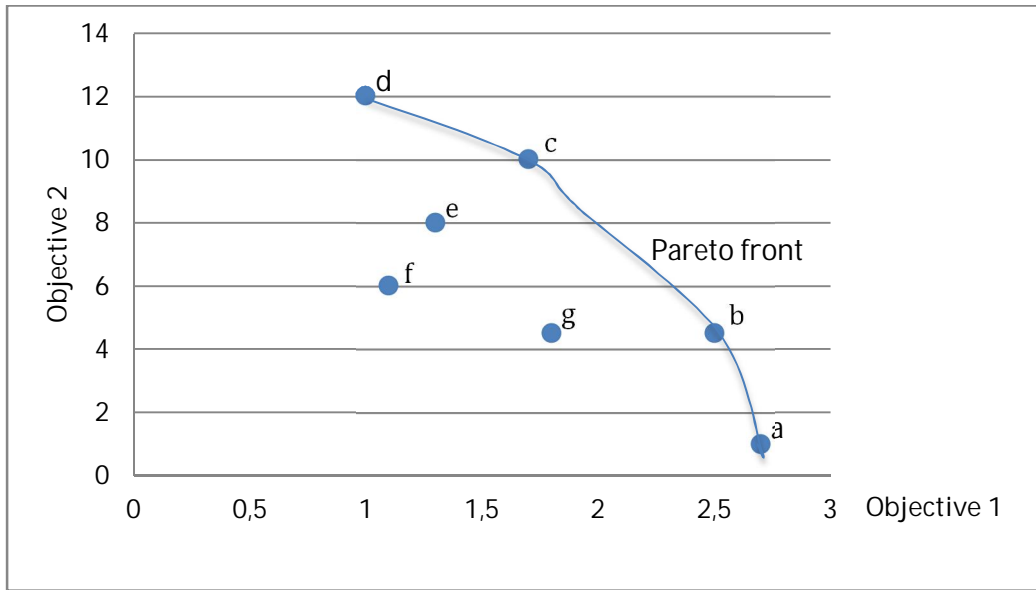


Figure 5. Example of Pareto optimal solutions (maximizing objectives)

Pareto optimality can also be used as useful analysis tool [Harman, 2007]. For example, the search may yield solutions, where a small change in one objective decreases the other objective in large amount. These kinds of solutions can be further examined to understand what factors have influenced the solutions.

2.3 Constraint Satisfaction Techniques

In artificial intelligence, constraint satisfaction has been widely researched for many years [Bartrak, 1999]. It has already proven to be useful for solving complex combinatorial problems in decision making processes. To solve a problem using constraint satisfaction, the problem has to be first formulated as a *constraint satisfaction problem* (CSP). Many problems from different areas, such as computer science, operation research and engineering can be modeled as CSPs [Tsang, 1995].

A CSP can be defined as a set of N variables, v_1, v_2, \dots, v_n , that can take values from predefined domains d_1, d_2, \dots, d_n and upon which constraints are defined. The constraints restrict the association of values to variables. A solution to a constraint satisfaction problem is an assignment of a value to each variable from its domain, such that all the constraints are satisfied [Tsang, 1995].

To illustrate the concepts of CSP, let us consider the problem of placing apples, oranges and mangoes in the cubicle shown in Figure 6. The problem is to fill all the regions (i.e., r_1, \dots, r_7) with apples, oranges, and mangoes such that no two adjacent regions have same fruit. One possible formulation of CSP representation:

- variables: each region can be a variable

Variables: r_1, r_2, \dots, r_7

- domain: the fruits to be placed will be the domains of the variables
Domain for each variable, i.e., each region: {apple, orange, mango}

- constraints: adjacent regions should have different fruits

$r1 \neq r2, r1 \neq r3, r2 \neq r3, r2 \neq r4, r3 \neq r5, r3 \neq r6, r3 \neq r4, r4 \neq r6, r4 \neq r7, r5 \neq r6, r5 \neq r7, r6 \neq r7$

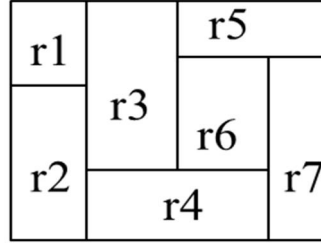


Figure 6. Placing apples, oranges and mangoes in a cubicle

CSPs are generally solved using exhaustive search with backtracking algorithms [Barreto et al., 2008]. When a variable is assigned a value, all the constraints involving this variable are examined. For example, if apple is placed in $r1$, then apple is removed from the neighbors of $r1$, i.e., from $r2$ and $r3$. This is known as constraint propagation, as constraint reduces the domains of the related variables. During constraint propagation, if no values in the domain can be assigned to the variable without breaking a constraint, the algorithm backtracks to the previous variable and tries the next value available in the domain. The search continues until a solution is found or all assignment alternatives are exhausted.

In many practical problems, it is often necessary to find not only a solution that satisfies the constraints, but to find a solution that also optimizes certain application dependent objectives. To solve these kinds of problems, a CSP is extended with an objective function that associates cost to a solution: these types of problems are called *constraint satisfaction optimization problems* (CSOP) [Tsang, 1995]. The goal is to find a solution that satisfies all constraints and either maximizes or minimizes the objective function. A CSOP consists of a CSP and an objective function, which is used to find a solution that optimizes certain application dependent characteristics. These problems are most widely solved using *branch and bound algorithm* [Bartrak, 1999]. The algorithm navigates the search space through backtracking and every time a partial solution is found, its cost is calculated and compared to the cost of the best solution found so far. If the partial solution cannot improve the current best one (that is, if its cost is higher than the current best solution's), the search branch is abandoned and the algorithm backtracks to the previous variable.

Solving a problem modeled as CSP can yield several kinds of answers: whether the problem has a solution or not (i.e., feasible or infeasible problem), or the problem has multiple possible solutions, or an optimal solution to the problem with respect to some quality measure. In addition to constraint propagation algorithms, *constraint solvers* can be employed for solving CSPs. Constraint solvers are programmable systems for efficiently removing values from the domain of the variables to reduce the portions of search space, which cannot possibly contain a solution to the problem [Jussien, 2005]. Constraint solvers provide a set of classes and already implemented algorithms for determining and resolving the proposed CSP automatically. A constraint solver provides a modeling language for expressing the CSP and contains different constraint

propagation algorithms for solving the model and for obtaining one or many solutions to the problem.

2.4 Software Architecture Design

Software architecture design has been traditionally considered as highly creative work, requiring special experience, assessment and talent. The latest IEEE standard 42010:2011 [IEEE, 2011] defines software architecture as “*a concept of a system in one’s mind or is a perception of the properties of a system*”. So, the architecture can be an abstract thing and has to be translated into an artifact to be presented to the stakeholders who have interest in the system. For this purpose, the standard defines the term *architecture description* for expressing the architecture of a system in the real world. The architecture description defines the environment, the elements that make up the system and their relationships. Also, it specifies the rules and constraints the system must satisfy and identifies the stakeholders of the system.

The stakeholders of a system have concerns with respect to the system of interest [IEEE, 2011]. For example, the stakeholders of a system can be developers, end users and project managers who have interest in the system. Each stakeholder can have one or more concerns related to the system. The software quality attributes are one such concern, which includes many attributes like modifiability, maintainability, performance, usability, portability, reliability, etc. [Bass et al., 1998]. The architecture should be designed so that it delivers the quality attributes specified by the stakeholders. In addition to the quality attributes, technical environment and organizational structure also have an influence on the software architecture. The focus in this thesis is to design software architecture only from the viewpoint of quality attributes and organizational structure, and the other drivers, such as technical environment or different stakeholders concerns are not considered in this research.

Software architecture design is often seen as an iterative process. The requirements of the system are used for creating the software architecture. Requirements state *what* the system is supposed to do and the software architecture defines *how* this could be achieved. There exist various methods for designing software architectures [Bass et al., 1998; Bosch, 2000], where each method has its own methodology for designing the software architecture. No matter what methodology is used, software architecture design process can be summarized as shown in Figure 7 [Jansen, 2008].

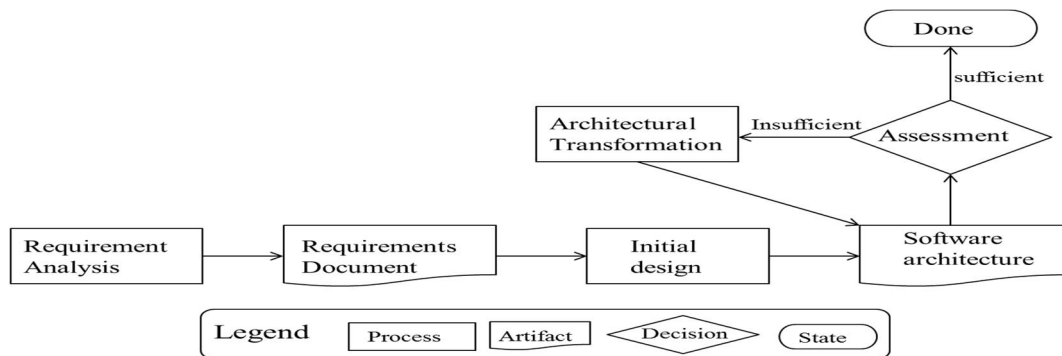


Figure 7. An abstract view on the software architecture design process [Jansen, 2008]

The architecture design process starts with gathering of requirements. In the initial design phase, the software architecture that satisfies some (parts of) requirements stated

in requirements document is created. After the initial design phase, the architecture is evaluated with respect to quality requirements. If the quality of the architecture is not sufficient, then the architecture is modified. The architect modifies the architecture by employing either a set of tactics [Bass et al., 1998] or architecture styles or patterns [Bosch, 2000; Gamma et al., 1994] for improving the design. These decisions eventually lead to the software architecture of the system. The process of evaluating the design and modifying it is repeated until a satisfied architecture is achieved.

To describe the software architecture of a system, different forms of communication are used. The most commonly used forms are natural language, templates, diagrams, pictures and formal language [Jansen, 2008]. Diagrams are very popular form for expressing complex relationships between the concepts used in the software architectures. For example, UML models [UML, 2015] are popularly used for modeling architectural concepts. In this thesis, we used UML class diagrams for expressing software architectures. Moreover, UML use case and sequence diagrams are used for gathering the requirements of the system.

2.5 Software Design Patterns and Architectural Styles

Software architectures and their design have been studied for decades. Over the years, a set of reusable solutions that have proven useful in solving certain design problems are identified. These reusable solutions are documented in form of design patterns or architecture styles. Gamma et al. [1995] described a list of design patterns for object oriented design. Shaw and Garlan [1996] reported a set of proven solutions as architectural styles. Similarly, good practices have been reported for enterprise systems [Hohpe et al., 2004] and service oriented systems [SOA, 2015].

A pattern can be any solution that is applied to solve a recurring problem in a specific context. It describes the problem, the solution, conditions for applying the solution and the consequences of the pattern. Design patterns are used to solve a problem at a subsystem level in the architecture, whereas architecture styles are used to solve a problem regarding the whole system architecture. The design patterns used in this work are Façade, Adapter, Strategy, Template Method and Mediator [Gamma et al., 1995]. The architecture styles used in this work are Client-Server and Message Dispatcher [Shaw and Garlan, 1996; Buschmann et al., 1996]. All these design patterns and architecture styles improve the modifiability of the system, but with a cost in efficiency. From here on, they are collectively called as *patterns*.

The *Façade* pattern wraps a complicated subsystem with a unified interface. This will make the subsystem easier to use through the higher-level interface defined by the Façade. The application of Façade increases the maintainability of the system. Moreover, structuring a system into subsystems reduces complexity, and a common design goal is to minimize dependencies between the defined systems.

The *Adapter* pattern wraps the interface of an existing class. It allows the changing of the interface of a service provider without affecting the clients, by introducing a wrapper class between the client and the service provider.

The *Strategy* pattern defines a family of algorithms, encapsulates each one, thus making them interchangeable. It allows varying the algorithm independently of the using client.

A Strategy pattern can be applied, e.g., when different variants of an algorithm need to be defined or there are several related classes that only differ in their behavior.

The *Template Method* defines the skeleton of an algorithm in an operation, deferring some steps to client subclasses. It allows subclasses to redefine these sub steps for providing varying functionality without changing the algorithm's structure.

The *Mediator* defines an object that encapsulates how a set of objects interact, thus promotes loose coupling by keeping objects from referring to each other explicitly. This pattern can be used when components are not able to interact directly with each other.

In the *Client-Server* pattern, a server component provides services to multiple clients. Client components request the server for the services they need. The servers are usually idle and wait for the requests from the clients. In cases of systems with large databases, the Client-Server architecture style is often used; the database is usually located in a server and clients request the data from the server.

Message Dispatcher pattern is used when a group of components needs to communicate with each other. It uses a centered Message Dispatcher for communication. All the components have a common interface that contains all the operations that are needed in order to send and receive messages to and from the dispatcher. In this style, components communicate with each other through a dispatcher. Although the components communicate with each other, they do not need to know where its messages will end up or from where it has received the messages.

Moreover, the message-based communication also acts as a powerful decoupling pattern in the system integration [Shaw and Garlan, 1996]. For example, the usage of messaging between the components reduces the mutual dependencies between the systems, as the dependencies are established between message endpoints, but not with statically defined component interfaces [Buschmann et al., 1996]. Similarly, the use of well-defined, documented and properly maintained interfaces can reduce the coupling between the work units. In this thesis, these solutions that reduce the coupling between components are termed as *decoupling solutions*.

2.6 Software Project Planning

Software project management contains several activities to ensure that the project is developed within available budget and schedule. The project management typically involves creating project plans and targets, selection and allocation of people, defining the development process, measuring and controlling the project progress [Albert and Dieter Rombach, 2003]. The creation of project plans and targets is known as *project planning*. Project planning identifies the goals, success criteria, and the resources needed for developing the project. The selection and allocation of people involves setting up the development teams and the responsibilities of team members. The development process defines the set of principles, methods and tools used by the project members in developing the project. The project manager measures the ongoing project activities (tracking whether important dates are met or not) and takes corrective measures for developing the project in a timely manner.

Project planning is an important task for many software engineering activities from development to maintenance. In an ideal situation, the available resources are unlimited, goals are obvious and the project success criterion is liberal. As such an environment is

rare in software development, a plan is necessary for successfully carrying out the projects. Project planning starts with a set of requirements and finishes with schedule and resource allocation for implementing the product. Project planning requires estimation of effort, budget and resources required for implementing the product and is referred as *cost estimation* [Boehm, 2000]. A central aspect in planning projects is the distribution of work to available resources. The *work distribution* has crucial effect on project outcome, as it influences the communication needed between the teams for developing the project. Work distribution is a challenging activity in the best of circumstances, and becomes even more challenging in the context of GSD [Lamersdorf, 2012].

GSD has become a major trend in software engineering. It is usually characterized by engagements with different national and organizational cultures in different geographic locations and time zones, using various traditional and IT-enabled means to collaborate [Hossain et al., 2011]. For example, a software product might be designed in Finland, while development is simultaneously carried out in different locations, such as India, Philippines and China. Finally, the product is integrated and tested at the Finnish site. This kind of development has become a common practice in software industry. The main motivation factors for companies to adopt GSD are: to have access to low cost personnel, to be close to a local market, increase the overall productivity (due to follow-the-sun development) and have access to a talented pool of developers worldwide. Moreover, the mergers and acquisitions of other companies also have driven companies towards GSD [Carmel and Tjia, 2005].

Although the advances in communication technology (e.g., video conferencing, shared repositories) made it possible to develop software globally, the division of software development work to distributed teams has created new challenges. Many of them are related to the communication, which cannot be completely solved using technology, as the degree of communication drops significantly if two persons are more than 30 meters away [Allen, 1977]. This is especially true for GSD, as development teams reside in different countries, work in different time-zones and speak different languages. The geographical distances between the teams limit the possibility of having face to face communication, the time-zone distances limit the team members from having real-time communication and socio-cultural differences between the teams cause misunderstandings [Ågerfalk et al., 2005; Hossain et al., 2011]. These communication barriers resulted from the inherent nature of GSD must be taken into account in assigning work to globally distributed teams.

Furthermore, the chosen software architecture may ease or hamper the distributed development [Clerc et al., 2005, Ali et al., 2010]. The software architecture of the system imposes the communication and coordination effort required for realizing the system [Arvritzer et al., 2008, Salger, 2009]. For instance, if two work units have high dependencies with each other, then high communication is needed between the teams to realize those work units. Thus, the software architecture of the system has to be taken into account in assigning work to distributed teams: whether to assign two highly coupled work units to the same team or to teams with medium or large communication barriers between them. Also, the decoupling solutions that can influence coupling between the work units are important. If work units are loosely coupled with each other, then each work unit can be assigned to one distributed team. In this situation, the coordination among teams is not an issue and the teams can independently develop the work units [Ali et al., 2010].

PART II – GENETIC ALGORITHMS FOR SOFTWARE DESIGN AND PROJECT PLANNING

This part describes the application of genetic algorithms for software architecture design and project planning. First, a method to apply genetic algorithms for software architecture design is presented. Second, the applications of genetic algorithms for project planning are presented. Finally, research conducted on the algorithms for software architecture synthesis is discussed.

3 GENETIC ALGORITHMS FOR SOFTWARE ARCHITECTURE DESIGN

This chapter describes the application of genetic algorithms for software architecture design. The approach was first presented in publication [P1] and has been used in other studies reported in publications [P2] and [P3]. It acts as a starting point for the thesis and corresponds to the thesis contribution 1, i.e., “using a set of UML diagrams for expressing the input for genetic architecture synthesis”. This chapter is organized as follows. First, Section 3.1 presents an approach where UML diagrams, such as use case, sequence and class diagrams are used for expressing the input required for genetic architecture synthesis. The approach is illustrated by designing the software architecture of an example system in Section 3.2. Finally, the chapter is concluded with a summary of the lessons learned.

3.1 Software Architecture Design using Genetic Algorithms

The application of genetic algorithms for software architecture design was originally envisioned and proposed by R  ih   et al. [2008]. The approach takes the functional and quality requirements of a system as input and uses the genetic algorithm to discover the architecture design decisions required for fulfilling the quality requirements of the system. The architecture design decisions are related to design patterns [Gamma et al., 1995] and architectural styles [Shaw and Garlan, 1996] applied in the context of the system. The architecture design decisions are applied as the mutations of the genetic algorithm and the fitness function is used for evaluating the quality of the architecture proposals. As a result, the genetic algorithm produces a set of architecture proposals. In this thesis, this approach has been integrated into a UML-based software design process, where UML diagrams such as use case, sequence and class diagrams are used for expressing the input required for the genetic architecture synthesis.

An overview of the approach is presented in Figure 8. The approach takes basic functional decomposition and operation characteristics of the target system (called *initial design* from hereafter) and quality requirements of the system as input. The quality requirements are given as weights of different sub-fitness functions used in the fitness function. In creating the initial design different UML diagrams are used (described in detail in Subsection 3.1.1). The initial design contains the logical classes (i.e., components) and the relationships between the classes required to fulfill the system functionality. As requirements gathering is always a manual task, we expect the user or architect to give the initial design and the quality requirements of the system. The initial design is used to generate an initial population of chromosomes, i.e., architecture proposals (described in detail in Subsection 3.1.2). The initial population is then evolved through mutations and crossover operations. The mutations introduce or remove patterns from the pattern base to (or from) the individuals during the evolutionary process.

Moreover, as a result of crossover new individuals are introduced into the population. Next, the individuals are evaluated for the quality requirements using the fitness function. The individuals for the next generation are selected following a fitness-proportionate selection method, as discussed in Subsection 2.1. The new population will then go through as many generations of crossover, mutations and selections as is needed until a terminating condition is reached. As a result, the genetic algorithm creates a set of architecture proposals, i.e., modifies the initial design in response to quality requirements of the system. The architecture of best individual is treated as the best architecture proposal produced by the genetic algorithm.

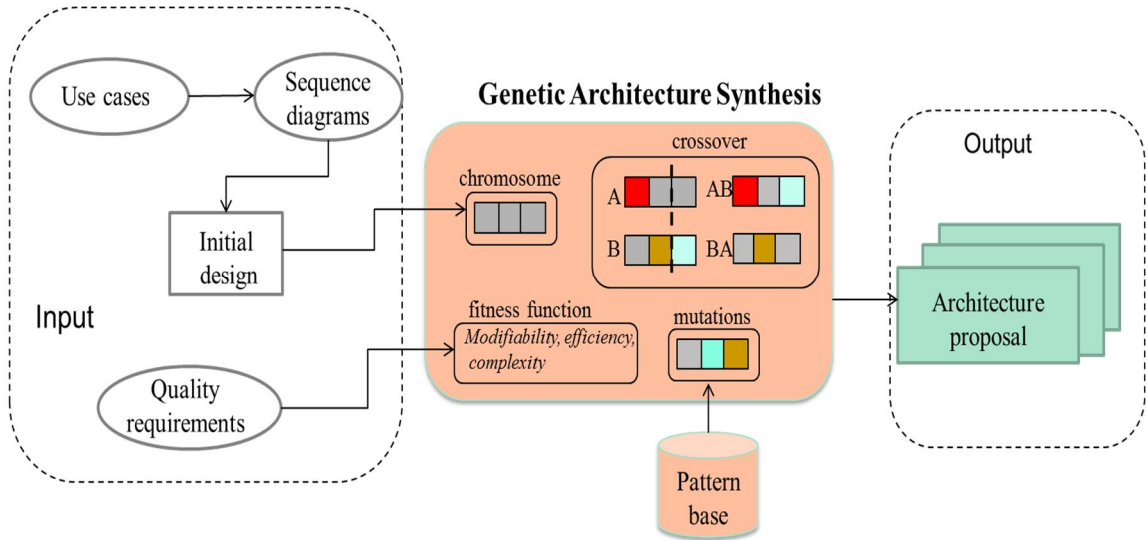


Figure 8. UML-based design process for applying genetic algorithm for software architecture design

3.1.1 Input

The input consists of initial design and quality requirements of the target system. The process of creating the initial design starts with gathering abstract requirements in a use case diagram. Next, the use cases are refined into sequence of messages in a sequence diagram representing the interaction between major components required for fulfilling the use cases [Selonen et al., 2001b]. The information in the sequence diagrams can then be used to construct the initial design of the system. The classes of the initial design correspond to the participants in the sequence diagram, the operations of the classes correspond to the incoming call messages of the participants of that class, and the dependency relationships between the classes are inferred from the call relationships of the participants.

In addition to simply expressing functional requirements as a set of classes and their relationships, the operations of the classes can be evaluated for characteristics, such as sensitiveness to variation (called hereafter *variability*) during the evolution of the system, frequency of use and parameter size. These values specify information about the nature of system and the needed support for evolving the system. The variability parameter indicates how likely the behavior of an operation will be changed in future. The frequency of use parameter indicates how frequently an operation will be. The parameter size indicates the number of parameters required for performing an operation. These parameters aid the genetic algorithm in achieving more accurate evaluation of the architecture's quality. The obtained initial design containing functional decomposition and operation characteristics of the system is then given as input to the genetic algorithm.

The quality requirements chosen to be optimized by the genetic algorithm are modifiability, efficiency and complexity. As the design patterns [Gamma et al., 1995] and architectural styles [Shaw and Garlan, 1996] applied in this work typically improve the modifiability of the system, modifiability is a natural choice. However, the introduction of the modifiability improving solutions negatively affects the efficiency of the architecture. Thus, as a counter attribute to modifiability, efficiency is used. In addition, to restrict the algorithm from applying patterns at every possible place, complexity is used as another counter argument to modifiability.

3.1.2 Encoding Initial Design into Chromosome

The information expressed in the initial design of the system must be encoded into a chromosome for the genetic algorithm to follow the biological analogies. The chromosome can be created by storing all information related to the operations in the initial design. However, it is difficult to store all information of an operation in a gene of the traditional chromosome, as each gene contains only one field. Therefore, to store all the information regarding an operation, a supergene [Amoui et al., 2006] is used. Unlike gene in the traditional chromosome, a supergene contains multiple fields to store data. For example, a chromosome containing two supergenes with 4 fields looks like Figure 9.

Supergene 1				Supergene 2			
field 1	field 2	field 3	field 4	field 1	field 2	field 3	field 4

Figure 9. Example of a chromosome with supergenes

In this work, a supergene is used for storing all the information corresponding to an operation. The number of operations in the initial design will be equal to the number of supergenes in the chromosome. Each supergene (shown in Figure 10) contains the following information related to an operation. Firstly, it contains basic information related to the operation itself. This includes its name, the operations that are invoking this gene's operation, type of the operation (i.e., attribute or operation), and additional support data. The support data include: the estimated frequency of use of the operation, the parameter size and variability of operation. Secondly, there is the structural information regarding the operation's place in the initial design: the class to which the operation belongs, the interface class the operation is part of, the dispatcher the operation uses for communicating with other operations, the other operations that call it through the dispatcher, and the design pattern it is a part of. All this information is stored in the supergene. The chromosome is formed by collecting all the supergenes, i.e., all the information related to all the operations. As the genetic algorithm works with a population of chromosomes, a single chromosome is not enough. In order to create an initial population the "base chromosome" is cloned multiple times and a mutation is randomly applied on each individual. Moreover, a special individual with no patterns is also added to the initial population. Thus, the initial population contains some diversity and has a better possibility of finding a good solution compared to the initial population with clones.

Name	Calling operations	Type	Call frequency	Parameter size	Variability	Class	Interface	Dispatcher	Dispatcher communications	Pattern
------	--------------------	------	----------------	----------------	-------------	-------	-----------	------------	---------------------------	---------

Figure 10. List of fields in a supergene

3.1.3 Mutations and Crossover

The mutations are applied at the supergene (i.e., operation) level, where a mutation results in adding or removing some information to (or from) a supergene. The mutations modify the architecture proposals by either introducing or removing a pattern to (or from) the architecture. The applied patterns are Message Dispatcher, Client-Server, Façade, Mediator, Strategy, Adapter and Template Method. Each mutation is by default targeted at one supergene, i.e., a mutation results in modifying some information of a supergene. However, the Message Dispatcher pattern is introduced in a slightly different way. In the case of Message Dispatcher, first a dummy supergene indicating the presence of Message Dispatcher is introduced into the chromosome. Next, operations can use the Message Dispatcher (i.e., by creating link to Message Dispatcher) for communication. Mutations are implemented as pairs of adding or removing patterns to (or from) the architecture. The applied mutations are shown in Table 4. In addition, the introduction and removal of an interface to and from the architecture is also applied as mutation. Also, each mutation (both addition and removal) is given a separate mutation probability.

Table 4. Applied mutations

<i>Mutations</i>
introduce/remove message dispatcher
create link/remove link to dispatcher
introduce/remove server
introduce/remove façade
introduce/remove mediator
introduce/remove strategy
introduce/remove adapter
introduce/remove template method
introduce/remove interface

The crossover is implemented as a traditional single point crossover, as discussed in Section 2.1. The crossover point is selected randomly. The crossover operation is also given a probability. In order to apply a mutation or crossover operator, first a supergene is randomly chosen. Next, the mutations or crossover operator that will be applied on the supergene is selected using a roulette wheel selection method (described in Section 2.1), where the size of the slice corresponds to the probability of applied mutations or crossover operation. The probabilities of applied mutations and crossover operator are adjustable parameters and can be tuned as desired. In addition, a null mutation, which does not do anything, is also included in the wheel.

After the application of a mutation or crossover operator, a corrective function is used for checking inconsistencies in the resultant chromosomes. For example, the crossover operation between parent 1 (where supergenes SG1 and SG2 are involved in pattern x)

and parent 2 (where supergenes SG1 and SG2 not involved in any pattern) may result in a chromosome, where SG1 is not involved with any pattern and SG2 is involved in pattern x. This chromosome does not belong to the solution space anymore, as it results in an invalid architecture. Thus, one of the supergenes must be modified to make the architecture valid again.

3.1.4 Fitness Function

The fitness function evaluates the quality of individuals. The fitness function measures the modifiability, efficiency and complexity of the architecture. Each individual quality attribute is measured using a sub-function, which can be either positive or negative depending on its implementation. Moreover, a weight is associated to each sub-function. The weights are used for emphasizing a certain sub-function over other sub-functions. The sub-fitness functions described in [P1] were inspired by coupling and cohesion metrics of Chidamber and Kemerer [1994]. These metrics are further fine-tuned based on how they affect the quality attributes of the system. These metrics have both positive and negative effect on the modifiability and efficiency of the system. For example, loose coupling improves the modifiability of the system, but strong coupling makes it difficult for future modifications. Therefore, to achieve clear sub-fitnesses, both the positive and negative effect of the metrics on modifiability and efficiency are taken into account. The fitness function $f(x)$ for a chromosome x is shown in equation (3).

$$f(x) = w_1 * sf_1 - w_2 * sf_2 + w_3 * sf_3 - w_4 * sf_4 + w_5 * sf_5 \quad (3)$$

where w_i is the weight for the respective sub-fitness sf_i and sf_1 measures positive modifiability, sf_2 measures negative modifiability, sf_3 measures positive efficiency, sf_4 measures negative efficiency and sf_5 measures complexity.

The positive modifiability rewards the use of interfaces, Message Dispatcher and Client-Server architecture styles based on their ability to change the system without disturbing the rest of the system. As an operation with high variability implies its tendency to change in future, the variability of the operation is considered in calculating the reward for using Message Dispatcher as well as Client-Server architecture style. The negative modifiability penalizes the direct calls, which creates strong coupling among the operations and their hosting components.

The efficiency is measured based on the cohesion characteristics of the architecture. The positive efficiency rewards highly cohesive classes, where many invocations of the operations are within the classes. This promotes the situations where multiple operations of one class are invoked by another class or vice-versa. As calls through the network are slow, the negative efficiency penalizes the calls made through Client-Server and Message Dispatcher. The frequency of calls is used in further emphasizing the penalty.

The application of mutations results in the heavy fragmentation of classes, i.e., increasing the number of interfaces and classes in the architecture. The complexity fitness function is used to limit the amount of classes and interfaces in the architecture. It is measured by counting the number of classes and interfaces introduced into the architecture.

3.1.5 Selection

After evaluating all the individuals in a generation, they are ordered according to their fitness values. The individuals with high fitness (i.e., the elite) are moved directly to the

next generation. Next, roulette wheel selection (as discussed in Section 2.1) is made on other individuals in the population. To apply roulette wheel selection, the individuals are ranked according to their fitness values, where the individual with the highest fitness gets the topmost rank and the individual with the lowest fitness gets the bottom rank. The individuals are then assigned a slice in the roulette wheel according to their rank. Next, the wheel is spun and the selected individual is moved to the next generation. After this process, the wheel is again initialized based on the remaining individual's fitness values. The wheel is then spun for as many times as needed to fill the population size.

3.2 Application to an Example System

The genetic architecture synthesis approach is applied to design the architecture of an automatic chocolate vending machine (called hereafter ACVM). This is intended to give an idea about how to apply genetic architecture synthesis for designing the architecture of a system at hand. The first step is to identify the relevant functional requirements of the system. A use case diagram can identify the major requirements of the system. The use case diagram of ACVM is shown in Figure 11.

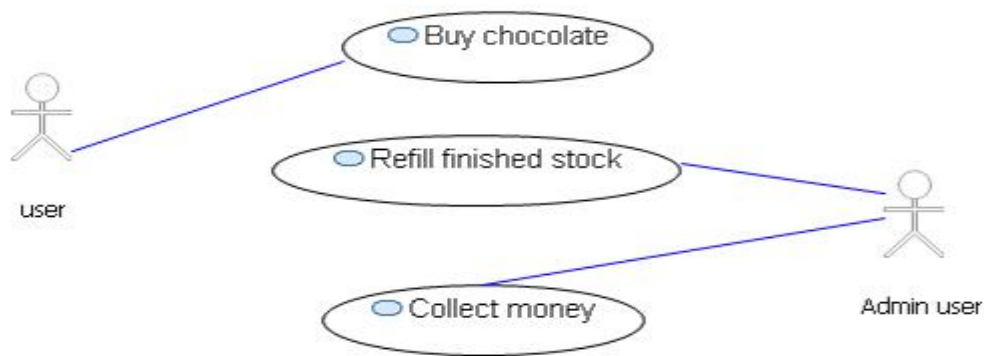


Figure 11. Abstract use cases of ACVM

The ACVM allows a user to buy desired chocolates and administrators can collect the money and refill the finished stock. The second step is to refine each use case into a sequence diagram. The second use case “refill finished stock” can be refined into a sequence diagram as shown in Figure 12.

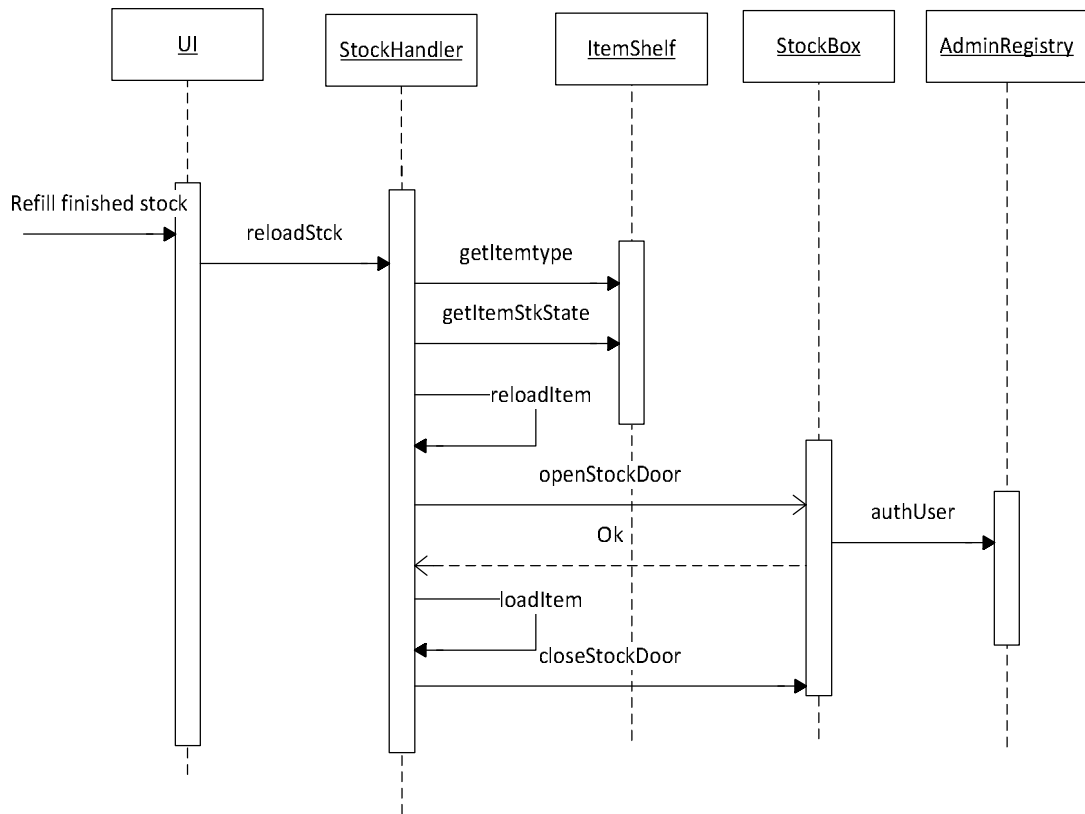


Figure 12. Sequence diagram for use case refill finished stock

The administrator should be provided a way to reload stock (reloadStck) from the user interface (UI). The reload stock will ask the administrator about what type of chocolate (getItemtype) to be reloaded and checks whether the selected chocolate item (getItemStkState) needs to be refilled. Next, the action reloadItem will start. As a consequence of that action, a call is made to open the stock door (openStockDoor) of the stock box. Before opening the stock door, the administrator is asked for authentication (authUser). If the authentication is successful, then the action load item (loaditem) will start. After loading the chocolates, the door of the stock box is closed (closeStockDoor). In addition to interactions between the operations, some characteristics of the operations can also be evaluated. For example, in future authentication mechanism between administrator and ACVM may be replaced with a new authentication mechanism. Taking this into account the “authUser” operation can be characterized as highly variable. Similarly, the operation “reloadItem” can be characterized as high frequency operation, as it is frequently called for reloading the stock.

As explained in Subsection 3.1.1, the information in the sequence diagram can be used to construct the initial design of the system. The participants in the sequence diagram (e.g. UI, ItemShelf and StockHandler) will be classes in the initial design and the incoming call messages (e.g. reloadStock, reloadItem and openStockDoor) will become operations. The corresponding fragment of initial design for the sequence diagram presented in Figure 12 is shown in Figure 13.

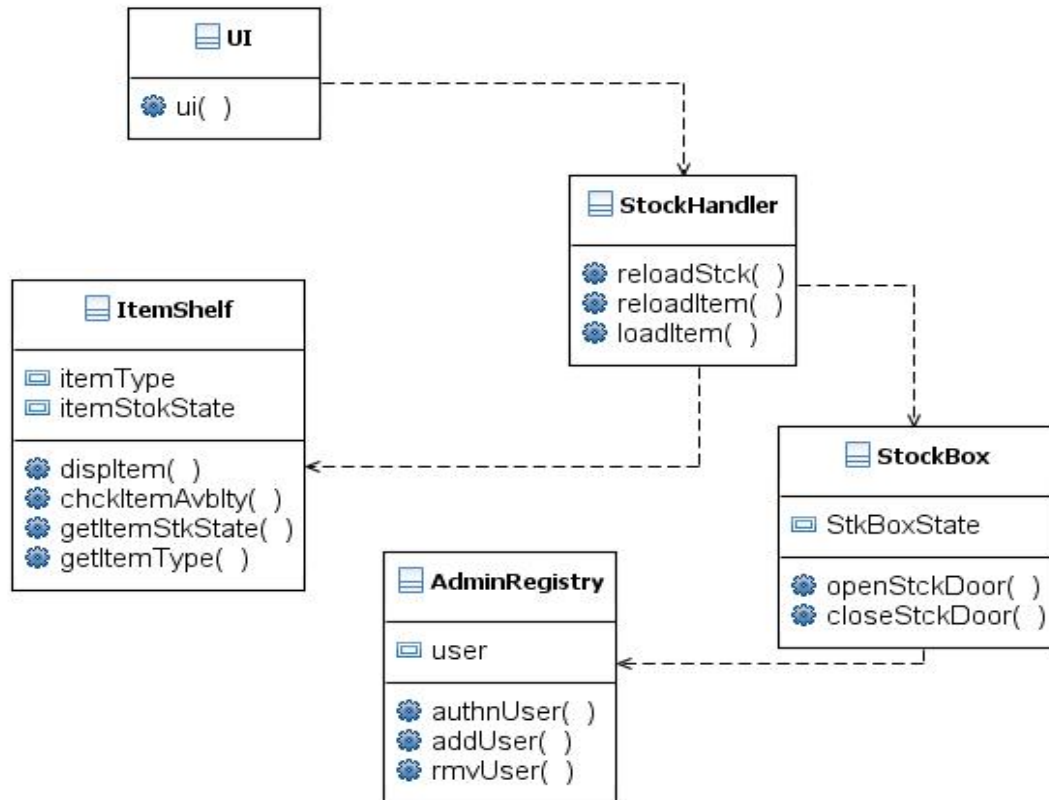


Figure 13. A fragment of initial design for ACVM

In the same way, the remaining use cases buying chocolate from the ACVM and collecting money from the ACVM can be refined into sequence diagrams. The participants of all the sequence diagrams and their corresponding relationships resulted in an initial design shown in Figure 14. As can be seen, the ACVM contains multiple components and multiple dependencies between them. The component UI (i.e., user interface) contains user interface controls for interacting with the ACVM. The components Button, Coin Handler and Selection interact with UI for obtaining the desired chocolate selection and for receiving the money from the user. The component AmountChecker checks whether the money entered by the user is sufficient for buying the chosen chocolate. If sufficient money is entered, then the component ItemShelf dispatches the selected chocolate. The components Storage and MoneyBox are used for storing the money entered by the users. The components StockHandler and StockBox are used for managing the stock. The component AdminRegistry lets the administrator to authenticate into the ACVM and collect money and manage stock.

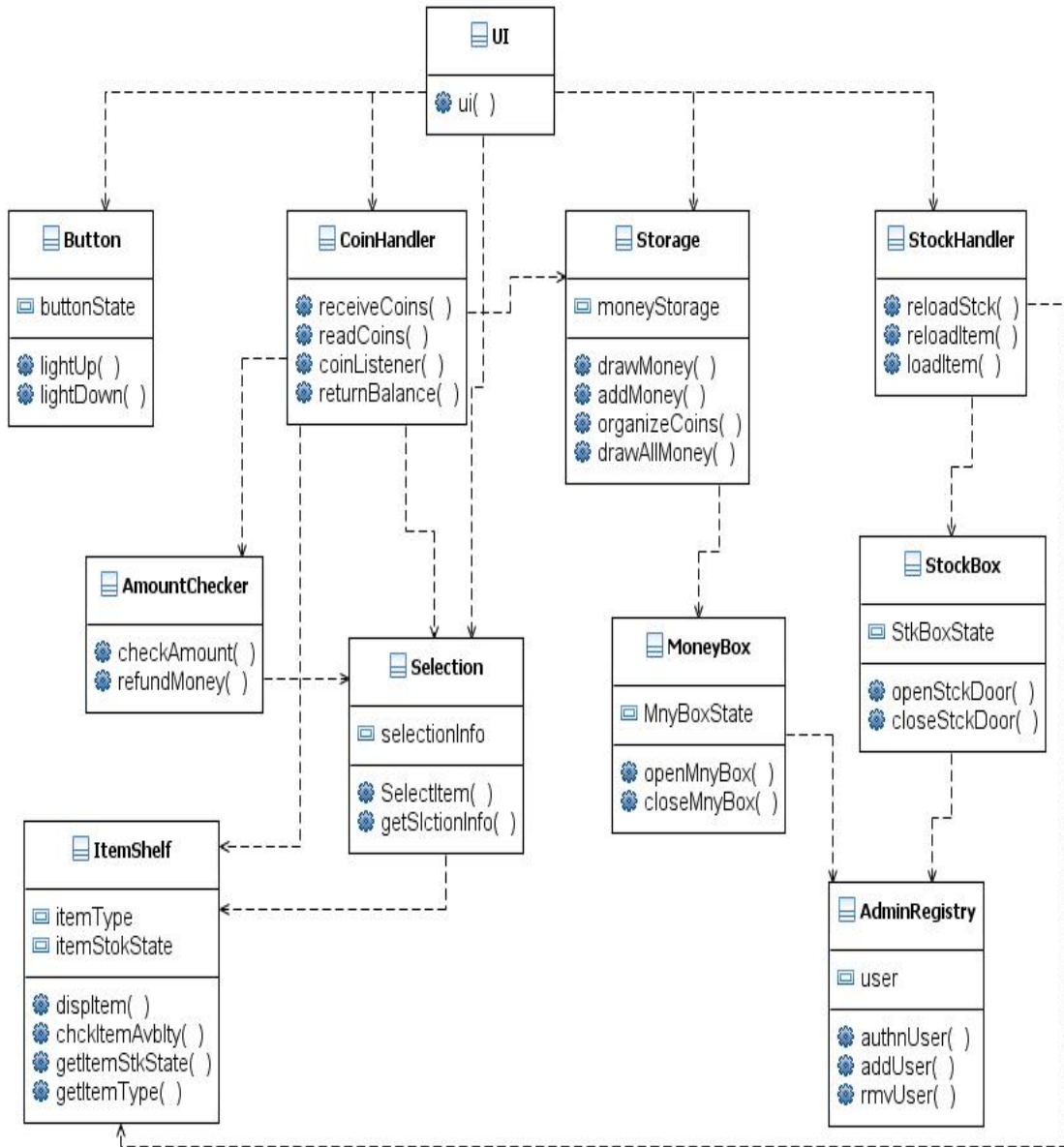


Figure 14. Initial design of ACVM

The genetic algorithm is applied on the ACVM with a population of 100 individuals for 250 generations. The mutation probabilities and fitness weights found after some experimentation are given. To obtain modifiable architectures, the modifiability sub-fitness function is emphasized over the efficiency sub-fitness function. The corresponding values are shown in Tables 1 and 2 of the Appendix A. The average fitness graph for 10 runs is shown in Figure 15. The fitness values of the individuals have improved after early generations, but after 200 generations the curve stabilizes and there is not very much improvement in the fitness values.

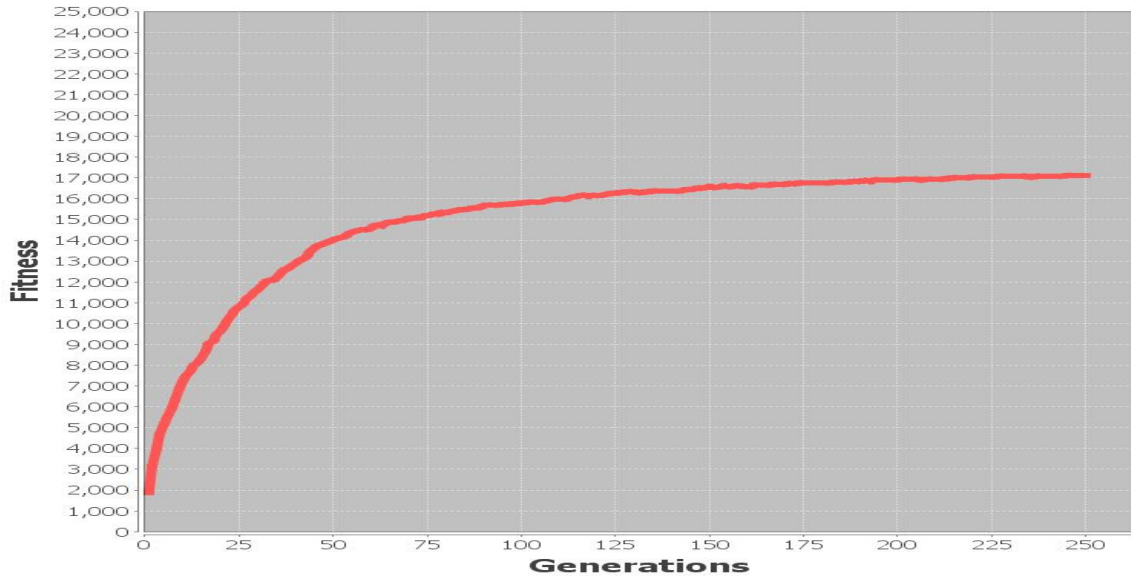


Figure 15. Fitness graph of ACVM

The architecture proposal suggested by the genetic algorithm is shown in Figure 16. Due to space limitations in the thesis, the architecture proposal is shown as a simplified class diagram associated with patterns chosen by the algorithm. To facilitate the presentation, the attributes and operations of the classes are not shown. For representing the application of Adapter pattern, a class corresponding to the name of the pattern is introduced between the classes that are using those patterns. For representing Strategy and Template Method, the operation on which corresponding pattern is applied is also shown. From here on, all the architectures described in this thesis are presented in a similar manner.

As can be seen from Figure 16, the proposed architecture proposal contains instances of Adapter, Strategy and Template Method Patterns. For instance, the application of Strategy pattern on reloadItem operation is sensible, as items can be reloaded in different ways. The genetic algorithm has proposed Template Method for organizeCoins operation. The application of Template Method is sensible, as coins can be organized differently that can be determined at development time. Although the genetic algorithm has produced sensible solutions in many cases, there are also some situations where the solutions proposed are not appropriate. For example, the Adapter pattern between UI and Storage is not appropriate, as it increases the time consumed to receive money from the ACVM. Moreover, in some situations, the genetic algorithm has produced a set of weird architecture proposals that no human architect would have done.

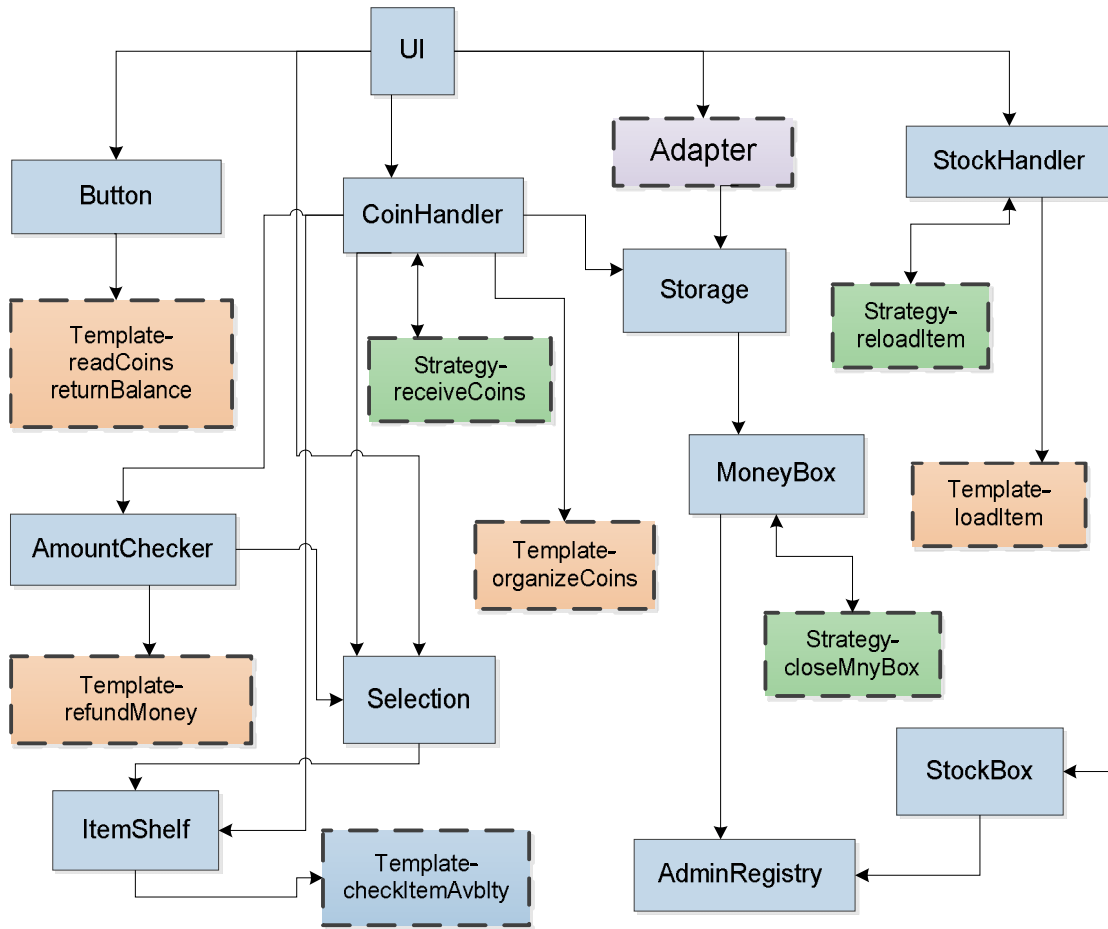


Figure 16. Proposal for ACVM

3.3 Summary

This chapter presents a UML-based design approach for applying genetic algorithms for architecture synthesis. The approach uses UML diagrams for expressing the input for genetic architecture synthesis. The use of well-known modeling language for expressing the input makes it easy for the practitioners, i.e., software architects to use the genetic architecture synthesis approach. The genetic architecture synthesis approach is still in its infancy and the limitations of the approach are discussed in Sections 9.1 and 9.2. We anticipate that if the patterns collection is significantly improved and the fitness function is tuned to consider the new design choices, then the approach could produce architecture proposals that are as good as the architecture proposals designed by experienced architects. In the current stage, the approach can be seen as a recommendation system, which can develop an initial architecture proposal of a system. The generated architecture proposal can be further improved using the architect's experience.

4 GENETIC ALGORITHMS FOR PLANNING GLOBAL SOFTWARE DEVELOPMENT PROJECTS

The objective of this chapter is to describe the application of genetic algorithms for planning GSD projects. This chapter summarizes the results of publications [P5] and [P6]. The genetic algorithm approach for planning GSD projects is first presented in publication [P5]. The approach has been further extended with Pareto optimality in publication [P6] to explore the solutions that balances the cost and completion time of the software project. The contents of this chapter correspond to the thesis contribution 3, i.e., “method for planning GSD projects using genetic algorithms” and thesis contribution 7, i.e., “method for multi-objective project planning using genetic algorithms”. This chapter is organized as follows. First, Section 4.1 discusses the meta-model used for allocating work to a GSD project. Section 4.2 shows how genetic algorithms can be applied to find (near-optimal) work assignment and schedule plans, together with decoupling solutions that support the chosen work distribution. Section 4.3 discusses multi-objective project planning using genetic algorithms. Finally, the chapter is summarized in Section 4.4.

4.1 GSD Work Distribution Meta-model

The overview of the work distribution meta-model is shown in Figure 17. The model contains two types of information: team characteristics (e.g. cost, skills and performance) and available task information. The model was derived in an iterative fashion by presenting main concepts identified in the literature to the research team and refining it according to the feedback from the team. In addition, the model was presented to two senior project managers in the industry and their feedback was taken into account in refining the model. The project managers were responsible for large and heavily distributed projects.

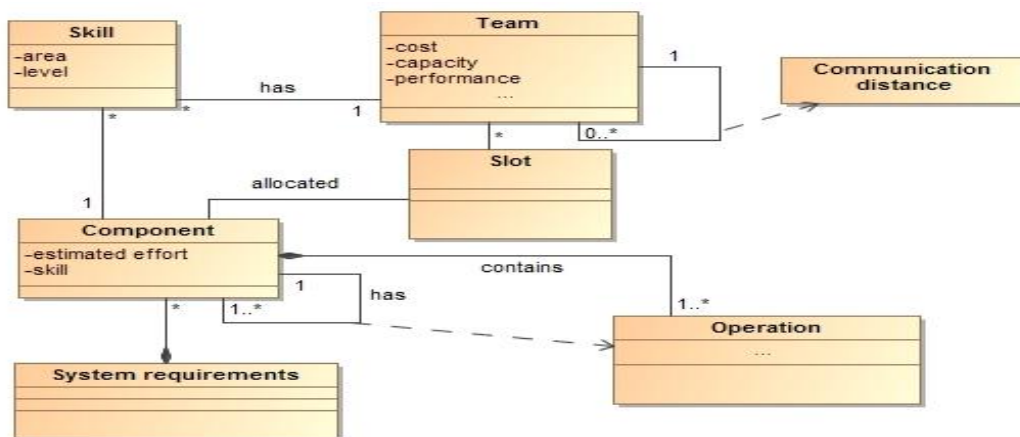


Figure 17. Overview of GSD work distribution meta-model

In the model, a team (i.e., a software development team) is an atomic resource available for developing the software. Each team has certain characteristics, for example, cost of the team per day, capacity, i.e., number of hours the team can spend on the project per day and performance of the team. These characteristics differ from team to team. Also, each team can have multiple skills with different experience levels. In addition, each team is associated with multiple slots, where each slot can be allocated to one task. The order in which tasks are allocated to slots determines the order in which team develops the allocated tasks. From here on this allocation order of tasks to slots is referred to as *team's development order*.

When a GSD project is distributed between two or more teams, each team has to communicate with other teams for implementing their tasks. As described in Section 2.6, the communication between the teams is constrained by the geographical, time-zone difference and socio-cultural distance between the teams. These distances that limit the communication between teams are used to estimate how far each team is located from another team from communication viewpoint. This distance is termed as *communication distance* and it specifies how easy or difficult it is to communicate with other teams in the project. It is expected that the project manager considers the characteristics of the teams, i.e., geographical, time-zone difference and socio-cultural distance between the teams in estimating the communication distance.

In the model, a component is a unit of work that can be allocated to a team. Each component hosts a set of logical operations, which will be implemented to realize the component. Moreover, each component is characterized in terms of the multiple set of skills required for implementing the component. In addition, the estimated effort, i.e., person-hours required for developing the component is also specified.

Each component can have one or more relationships with other components. Each relationship can be either a dependency relationship or precedence relationship. The dependencies between the components originate from the dependencies among the operations they host. For example, if an operation “x” of component C1 (shown in Figure 18) needs to communicate with operation “y” of component C2 to complete its action, then it means that component C1 depends on component C2. This implies that the team developing component C1 needs to communicate with the team developing component C2 for some information, such as the provided interface or required interface, etc. In addition, a component can have a precedence relationship with other components. The precedence relationship is used for expressing the preferred development order of the components. For example, a situation where component C2 uses large data entities produced by C1, or if component C2 can be sensibly tested only if component C1 exists, both these cases imply that component C1 precedes component C2, then developing C1 is a prerequisite for developing C2. Otherwise, the team developing C2 has to wait until C1 is developed. It is assumed that the architect determines the precedence relationships based on various facts known about the components.

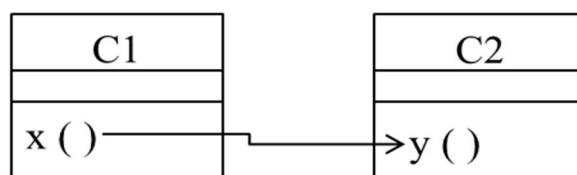


Figure 18. Dependency between components

4.2 Using Genetic Algorithm for Planning GSD projects

An overview of the procedure using genetic algorithms for planning GSD projects is presented in Figure 19. As input in the approach, the initial design (with available components), available teams and their characteristics are given. The cost and duration goals are expressed as the weights of the sub-fitness functions used in the fitness function. Based on the input, an initial work distribution model is created automatically by assigning each component of the target system to a randomly chosen development team at a random position (slot) in the team's development order. The initial work distribution model is used for constructing an initial population of work distribution proposals (discussed in detail in Subsection 4.2.1).

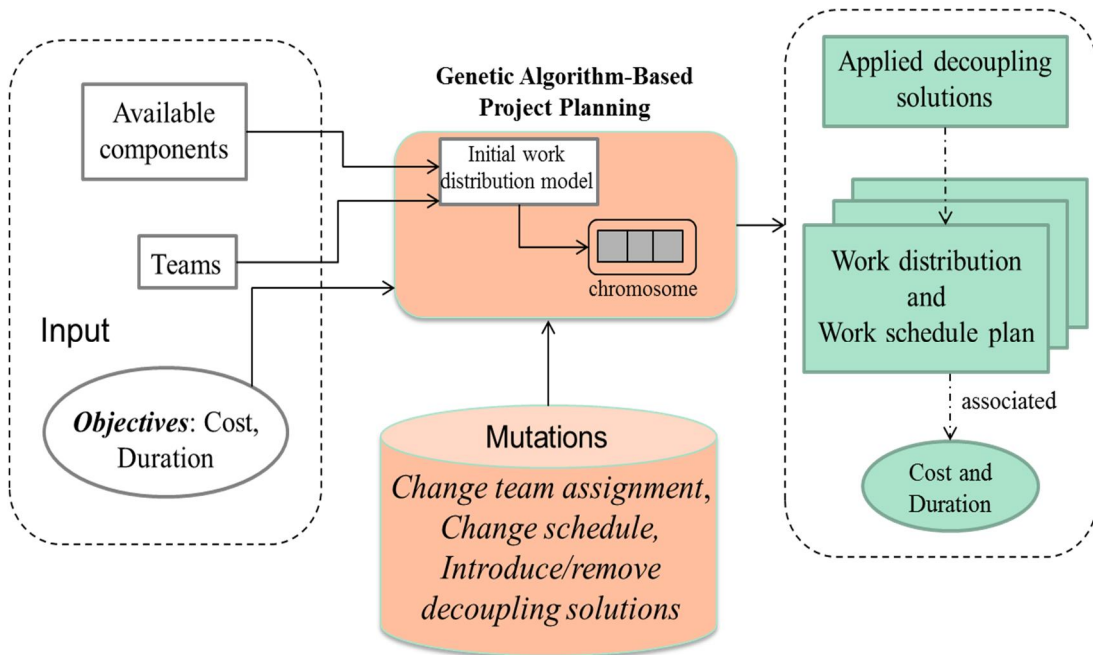


Figure 19. Overview of genetic algorithm approach for planning GSD projects

During the evolution, the application of mutations and crossover alter the population (i.e., work distribution models). The mutations change the assignment of a component from one team to another team and alter the position of the component in the team's development order. In addition, the introduction and removal of decoupling solutions between the components are also applied as mutations. The crossover operation is implemented using a traditional one-point crossover, as discussed in Section 2.1. In each generation, the fitness function evaluates the work assignment with respect to project goals, i.e., cost and duration required to implement the allocated work units. After evaluating the fitness, the individuals required for the next generation are selected using roulette wheel sampling, as discussed in Sections 2.1 and 3.1.5. This process continues until the specified terminating condition. As the result, the genetic algorithm produces a set of work distribution and schedule plans, together with the decoupling solutions introduced between the components.

In this work, two decoupling solutions are used. The focus here is not in studying the detailed effect of different decoupling solutions, but rather in the general idea of taking the decoupling solutions into account in the GSD work allocation problem. Therefore, as representative samples only messaging (representing strong decoupling) and interfaces (representing weak decoupling) are used.

4.2.1 Encoding Initial Work Distribution Model into Chromosome

To apply the genetic algorithm, the initial work distribution model must be encoded into a chromosome. The data related to all the available components in the initial work distribution model is collected for creating the chromosome. Since a gene in a traditional chromosome contains only one field, it is difficult to store all the data related to a component in one gene. Therefore, a supergene (discussed in Subsection 3.1.2) is used for storing the information regarding a single component. This supergene differs from the supergene described in Subsection 3.1.2 by taking into account the team information and is encoded at the component level. A supergene SG_i (shown in Figure 20) contains the following information related to a component. Firstly, the basic information about each component, such as the operations associated with the component, other components depending on this supergene's component, the components that it has precedence relationship with, and other attributes like component's name, estimated effort and required skills. Secondly, it contains information about the team name in the organization to which the component is assigned to and the position of the component in the team's development order. The development order defines the order in which teams develop their components. Thirdly, there is the information regarding the introduced decoupling solutions, such as the messaging solution it uses, components that communicate with it using the messages, and the interface used by it.

Associated operations	Depending components	Preceded components	Name	Effort	Skill	Team	Position	Messaging	Message communications	Interface
-----------------------	----------------------	---------------------	------	--------	-------	------	----------	-----------	------------------------	-----------

Figure 20. Structure of a supergene SG_i

The chromosome (see Figure 21) handled by the genetic algorithm is created by collecting all supergenes, i.e., all data regarding all components. The initial population is created by first cloning the chromosome multiple times and then random mutations are applied to ensure diversity in the initial population.

SG_1	SG_2	...	SG_{m-1}	SG_m
--------	--------	-----	------------	--------

Figure 21. Structure of a chromosome (collection of supergenes)

4.2.2 Mutations and Crossover

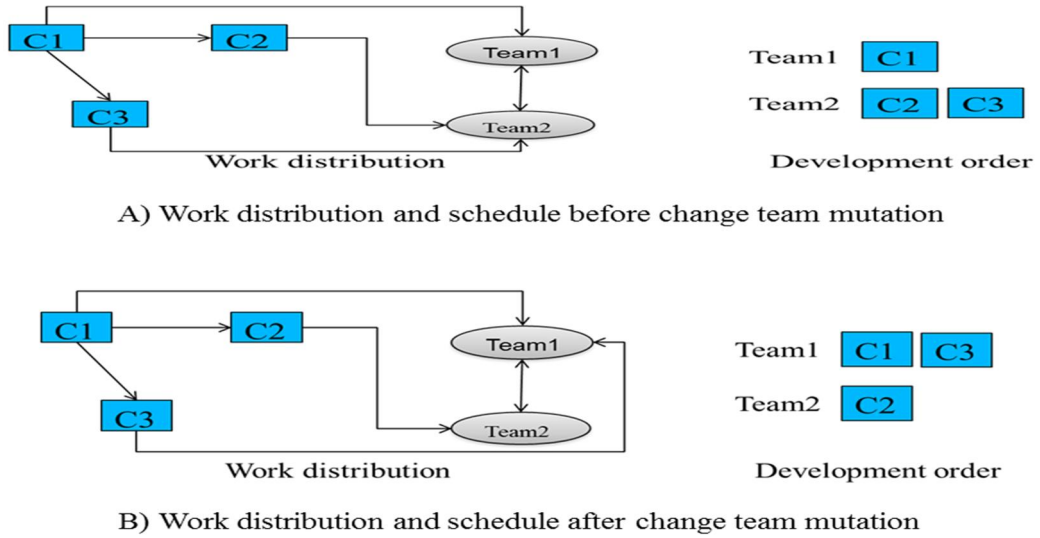
The mutations are applied to change the team assignment and to change the development order of the teams. Also, the mutations introduce or remove decoupling solutions (messaging and interfaces) between the components. The applied mutations are shown in Table 5. The mutations related to decoupling solutions are applied in the same way as described in Subsection 3.1.3. Here, they are applied only as a means to support work distribution between teams.

Table 5. Mutations applied for work planning

<i>Mutations</i>
change team
change development order
introduce/remove message dispatcher
create link/remove link to dispatcher
introduce/remove interface

The *change team* mutation replaces the team allocated for developing the component with another team. The new team is chosen randomly from the set of available teams. After the application of the mutation, the component's position in the newly chosen team is updated. It is given last position in the newly chosen team's development order. A prerequisite for applying change team mutation is that the team should have the necessary skills for developing the component.

To illustrate the application of change team mutation, let us consider an example work distribution and schedule shown in Figure 22 A. In the figure, applying change team mutation on component C3 may change the team assigned to component C3 from Team 2 to Team 1, as shown in Figure 22 B. Moreover, component C3's position in the newly chosen team is modified. As it is newly added to Team 1, it is given last position in the Team 1's development order.

**Figure 22.** Overview of change team mutation

The *change development order* mutation changes the order in which the components are developed by the team. It exchanges the position of the mutated component with the position of a randomly chosen component assigned to the team. For example, applying change development order mutation on component C3 in Figure 23 A can modify the development order as shown in Figure 23 B. However, the change development order

mutation is implemented only if the application of mutation does not violate the precedence constraints associated with the component.

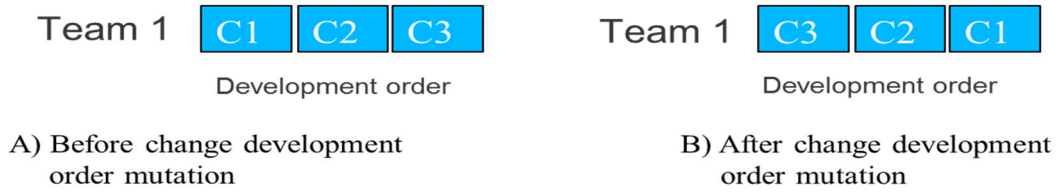


Figure 23. Overview of change development order mutation

The crossover operation is applied between supergenes and is implemented as a single-point crossover, as discussed in Section 2.1. After applying a mutation or crossover operation, the corrective function is used to ensure that the chromosome stays legal. For example, consider two parent chromosomes with a set of supergenes, as shown in Figure 24. If crossover happens between them at crossover index 2 as shown in the Figure 24, then it results in two child chromosomes, where child 1 has two supergenes SG2 and SG3 with same positions in the Team 1's development order and in child 2 supergene SG2 has position two in Team 1's development order, instead of position one. For making chromosomes legal again the corrective operation is used. In this study, the preconditions and a corrective function are used as opposed to discarding the illegal chromosomes from the population or punishing the illegal chromosome heavily (as discussed in Section 2.1) [Michalewicz, 1995]. The roulette wheel sampling is used for selecting the mutations and crossover operations, where the size of the slice corresponds to the probability of a mutation or crossover operation. In addition, a null mutation, which does not do anything, is also included in the wheel.

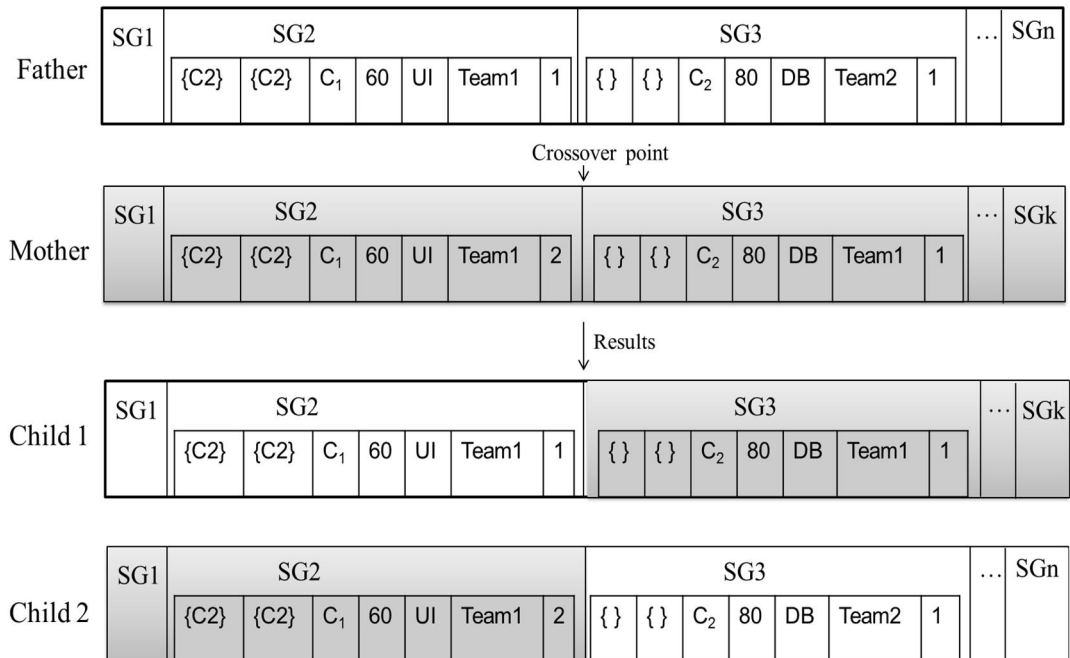


Figure 24. Overview of crossover operation

4.2.3 Fitness Function

The fitness function evaluates each chromosome with respect to cost and duration objectives. The fitness function $f_{wm}(x)$ for a chromosome x is shown in equation (4).

$$f_{wm}(x) = w_1 * Duration * \alpha + w_2 * Cost * \beta. \quad (4)$$

The fitness function consists of two sub-fitness functions duration and cost. The weight w_1 is used for emphasizing duration sub-fitness function and weight w_2 is used for emphasizing cost sub-fitness function. As cost and duration are of different units, the scale factors α and β are used to keep their magnitude in the same scale. The goal here is to find individuals, which minimize the fitness function $f_{wm}(x)$. This fitness function is an abstract function and a more fine-grained fitness function is needed for real world situations. This fitness function is capable of producing a single optimal solution. However, a more appropriate way to evaluate cost and duration sub-fitness functions would be using Pareto approach, as discussed in Section 4.3.

In this work, it is assumed that all the teams will be developing the work assigned to them in parallel. In this aspect, the duration of the project will be the maximum time spent by a team among all the available teams. Suppose that T_1, T_2, \dots, T_n are the set of teams used in the project, then the duration can be expressed as equation (5).

$$Duration = Max(Duration(T_1), Duration(T_2), \dots, Duration(T_n)). \quad (5)$$

The time spent by an individual team can be estimated by summing the duration of all the components that are assigned to the team. The duration spent by a team T_i on a project can be expressed as equation (6).

$$Duration(T_i) = \sum_{k=1}^C df(k) + wf(k) \quad (6)$$

where $k = 1, \dots, C$ and C is equal to the total number of components assigned to team T_i , function $df(k)$ specifies the duration of component k , and function $wf(k)$ measures whether the team has to wait for some time due to the precedence relationship of component k .

The duration required by a team for developing an individual component can be estimated by dividing the effort spent by the team on a component with the capacity of the team. The duration spent by team T_i for developing component k can be expressed as equation (7).

$$df(k) = \frac{effort(k) + cef(k)}{capacity(T_i) * performance(T_i)} \quad (7)$$

where $effort(k)$ represents the effort specified for component k , function $cef(k)$ represents the communication effort required for developing component k . The representations $capacity(T_i)$ and $performance(T_i)$ denotes the capacity and performance of team T_i .

The effort required for inter-team communication can be estimated using the communication distance between the teams and the coupling caused by the dependencies between the components. The communication effort required by a team in developing component k can be measured as equation (8).

$$cef(k) = \sum_j coupling(k, j) * dist(t(k), t(j)) \quad (8)$$

where j is a pointer sequentially traversing all the components that have a relationship with component k . Function $coupling(k, j)$ estimates the coupling cost between two inter-team components. The coupling cost is derived from the types of dependencies between the inter-team components. If the dependency is through messaging, then the coupling cost is assumed to be low. If the dependency is through interface the coupling cost is assumed to be medium and in the case of direct dependency the coupling cost is assumed to be high. Also, the additional effort required by the components in using the introduced decoupling solution is considered in specifying the coupling cost. Function $t(i)$ gives the team information of the i^{th} component. Function $dist(t(i), t(j))$ gives the communication distance between any two teams. The communication distance is estimated by the user using the scale {low, medium, high}. This value is converted into a corresponding relative numerical value.

In addition, the teams may need to wait for certain time due to the precedence relationships between components. The waiting time $wf(k)$ is zero if the preceded components have been already developed. Otherwise, the team has to wait until preceded components are developed.

The cost for developing the planned software can be estimated by summing the cost consumed for using each team. Suppose that T_1, T_2, \dots, T_n are the set of teams used in the project, then the cost can be expressed as equation (9).

$$Cost = \sum_{i=1}^n Duration(T_i) * cost(T_i) \quad (9)$$

where $Duration(T_i)$ specifies the duration spent by team T_i on the project and $cost(T_i)$ represents the cost of team T_i per day.

4.2.4 Experiments and Evaluation

The goal of the experiment was to study what kind of work distribution plans will be produced by the genetic algorithm when either cost or duration is emphasized over the other. This corresponds to a situation, where a project manager is exploring the possibilities to run the project either in shorter time with possibly more cost, or cheaper but with longer development time. For this experiment, the electronic home control system (given in Appendix B) is used as a target system. From hereafter, the electronic home control system is referred as ehome. The components that need to be developed for realizing ehome can be obtained by creating its initial design. The initial design of ehome resulted in 12 components, as shown in Figure 25. For presentation purpose only components (without operations) and their precedence relationships are shown (see Appendix B for details). Each component is further characterized according to the skills and effort required for developing the component. Moreover, eight precedence relationships are identified between the components. For example, the skill requirements for the CoffeeMachine component are experience in user interfaces and database development, and there should also be programming specialists. Also, the WaterControl component has precedence relationship with CoffeeMachine component.

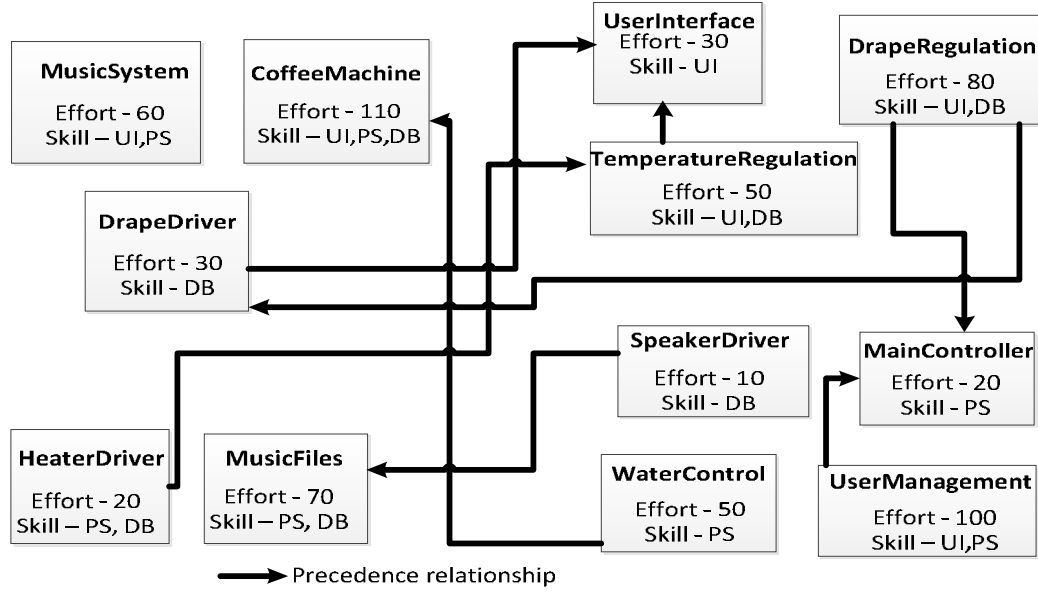


Figure 25. Ehome system with effort, skills and precedence relationships

The teams used for developing ehome are shown in Figure 26. In this experiment, the teams are assumed to have same performance in developing the project. Moreover, in this experiment, it is assumed that each team can spend only a limited amount of time on the project. This is expressed as project hours in the team structure shown in Figure 26.

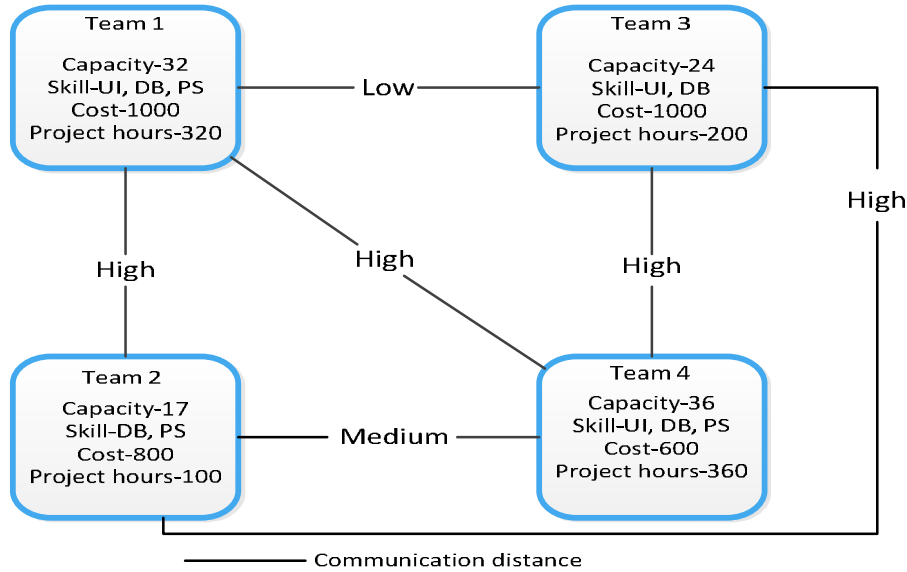


Figure 26. Team structure

The suitability of the genetic algorithm and the developed meta-model for planning GSD projects are examined by conducting two tests. The duration sub-fitness function is emphasized over cost sub-fitness function in the first test and cost sub-fitness function is emphasized over duration sub-fitness function in the second test. In both the tests, the genetic algorithm starts with an initial work distribution model created by randomly assigning components of ehome to teams shown in Figure 26. The best work distribution and schedule plans resulted from the tests is shown in Figure 27 and 28. For presentation purpose, simplified pictures showing only the team assignment, inter-team dependencies

and decoupling solutions introduced between the components are presented. The value under the name of the component specifies the position of the component in the development order of the team.

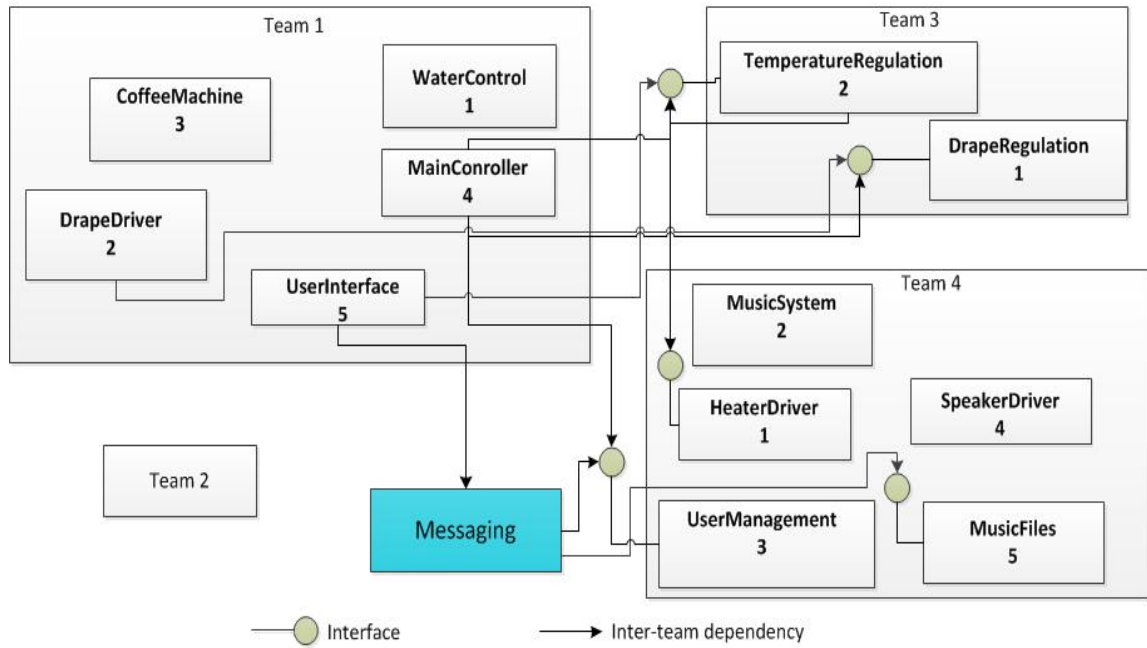


Figure 27. Best work distribution and schedule plan when duration is optimized

The results produced by the genetic algorithm were inspected for evaluating how the genetic algorithm has influenced the work distribution proposals. As can be seen from Figure 27, the genetic algorithm has introduced messaging and interfaces between the components and has also scheduled components according to their precedence constraints. As the objective in test 1 is to develop the project quickly, the genetic algorithm has placed components that require communication to teams with small distance and also introduced decoupling solutions between components assigned to teams with high distance. In particular, the application of messaging is appropriate. For example, messaging has been applied between UserInterface and MusicFiles components, which are assigned to Teams 1 and 4 that have high communication distance with each other. The introduction of decoupling solutions between components assigned to distant teams reduces the communication overhead between the teams, which reduce the overall duration. Moreover, the distribution of work to three available teams also decreases duration, as more hours are spent on the project per day. However, the cost of the planned software remains high, as two out of three teams (i.e., Team 1 and Team 3) assigned for the work are very expensive.

In the work distribution proposal shown in Figure 28, majority of work is assigned to low-cost teams and all the available teams are not used. For example, Team 3 which is very expensive is not assigned any work and Team 2 is used for developing only one component. The majority of work is moved to Team 4, which is cheap compared to other teams. Moving work to cheap teams reduces the cost of developing the planned software. In theory, the allocation of all components to the team with lowest cost results in work distribution with minimum cost, since this would also save effort and cost spent in communication. However, the genetic algorithm did not produce such a solution, as there is a constraint on the number of hours a team can spend on the project. Moreover,

as only two teams are doing the entire work, the duration required for developing the software increases.

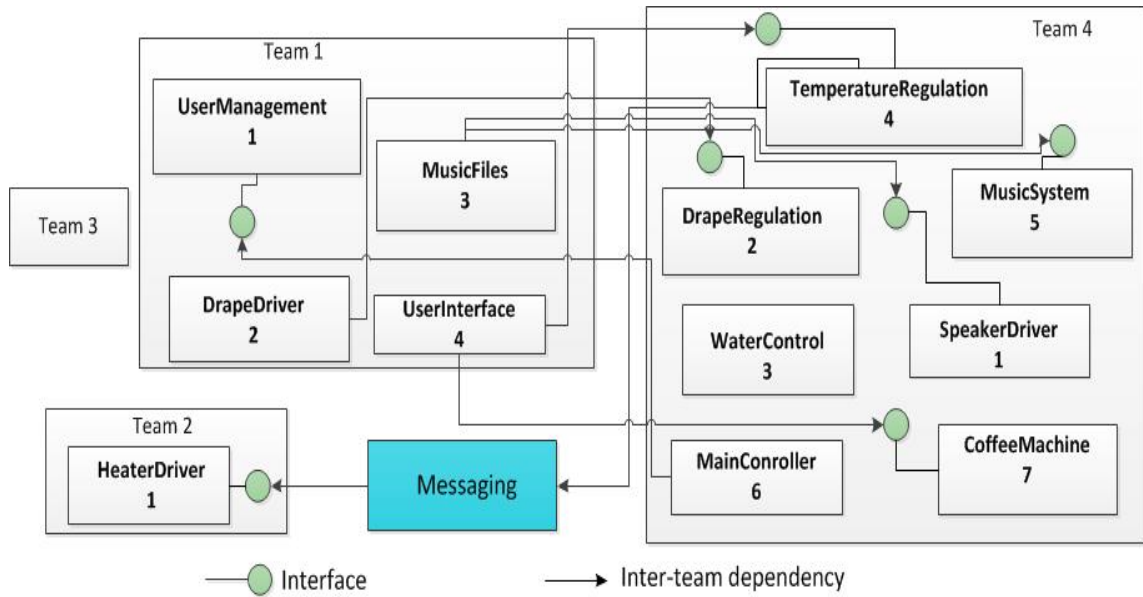


Figure 28. Best work distribution and schedule plan when cost is emphasized

In addition to resulted work distributions, the estimated cost and duration values of both the tests also indicated the conflicting nature of cost and duration. The average cost and duration values of both the tests for 10 runs are shown as boxplots in Figure 29. For test 1, the estimations show that duration can be reduced by relaxing the cost to develop the project. In the case of test 2, the estimations show that cost can be decreased by relaxing the duration of the project.

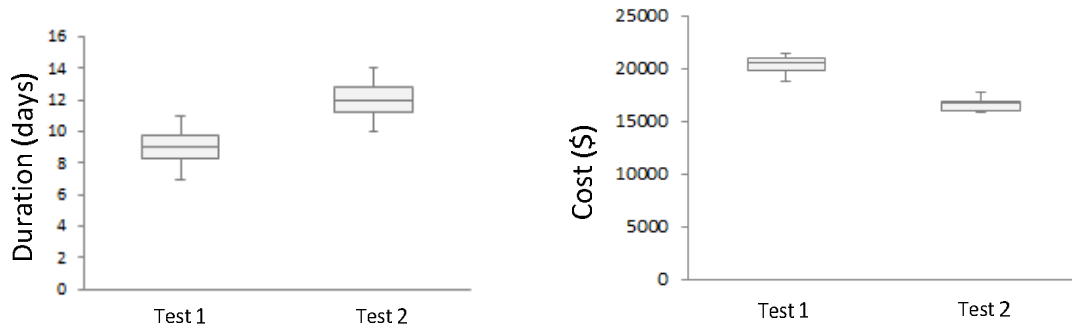


Figure 29. Estimated duration and cost for Test 1) duration is emphasized, and Test 2) cost is emphasized

The resulted work distribution plans and cost and duration estimations showed that the proposed genetic algorithm produces sensible results demonstrating the conflicting nature of cost and duration, i.e., duration required to realize the work plan can be decreased at the expense of cost and vice-versa. To reduce the duration of development, the genetic algorithm has moved the majority of components that need communication to same teams and proposed decoupling solutions between the components that are assigned to teams with high communication distance. This kind of work distribution is quite sensible, as decrease in communication between teams reduces the duration as well as the cost of development. In some cases, the genetic algorithm has also proposed inappropriate solutions. For example, in Figure 27, introducing messaging between

MainController and UserManagement classes could have further reduced the duration, but the genetic algorithm has not suggested that solution. Also, in some cases the genetic algorithm has suggested some entirely new and interesting work distribution proposals which a project manager would not make. However, using a single weighted fitness function has drawback that the optimization of one objective may be achieved at the expense of other objective. This limitation is eliminated by evaluating each of the objectives to be optimized as a separate goal in the next section.

4.3 Multi-Objective Project Planning with Genetic Algorithms

Software project planning is a multi-objective problem and it is usually not possible to find a single solution that is optimal with respect to both cost and duration of the project [Chang et al., 2001]. In such situations, instead of presenting one optimal solution to the project manager, it is more pragmatic to present a set of candidate solutions that are reasonably distributed along the possible objective values. To support manager in choosing solutions that satisfy different objectives, the project planning approach discussed in previous section has been extended to find the approximations of Pareto optimal front (discussed in Section 2.2). This allows the project manager to select an appropriate solution according to the project priorities.

4.3.1 Pareto Optimal Front

In order to compute Pareto front, the problem objectives need to be evaluated separately. In our case, this means evaluating the cost and duration of a solution individually. For calculating the duration fitness $df(x)$ and cost fitness $cf(x)$ of a solution x , the duration and cost sub-fitness functions presented in equations (5) and (9) are used. Now, each individual has two fitness values cost fitness and duration fitness, instead of just one fitness value. After calculating the fitness values, the individuals are sorted according to their duration fitness in the ascending order. Next, the individuals that form the Pareto front are selected. The first individual, i.e., the individual with lowest duration is included in the Pareto front, as it represents one of the extreme choices. The search starts with this individual and the sorted individuals are then iteratively processed until an individual with lower cost is found. If any such individual exists, then it is included in the Pareto front and the search is restarted from that individual. This process is repeated until all the individuals are explored. Since the individuals are sorted according to the duration fitness, the newly selected individuals will have higher duration than the best individual.

However, as each Pareto front usually contains few individuals, i.e., less than ten individuals and the population used for the genetic algorithm contains typically more than 100 individuals, choosing only one Pareto front is not enough to make a sufficient population. Thus, for having sufficient population, the Pareto selection is again repeated on individuals that do not participate in the Pareto front. This process is repeated until the population for the next generation has at least the number of required individuals.

4.3.2 Experiments and Evaluation

The main aim of the experiment was to study how the Pareto optimal genetic algorithm produces work distributions in a tradeoff between the cost and durations of the project. In the experiment, two questions were studied: 1) what are the advantages and disadvantages of using low-cost remote teams, as compared to teams located at same site

that have high-cost, and 2) how to choose a suitable development site in a tradeoff between developing the project quickly versus developing the project cheaply.

In this experiment, ehome system used in Subsection 4.2.4 has been used again. For simplifying the experiments, no decoupling solutions are used. Two tests are conducted to study question one. In the first test, all the teams available to develop the work are located at the same site (shown in Figure 30 A) and in the second test, all the teams chosen for development are at different sites (shown in Figure 30 B). In both of the tests, it is assumed that all the teams have same experience level. All the non-dominated solutions resulted from the Pareto fronts of both the tests are shown in Figure 31. The smaller circles denote the results of test 1 (i.e., teams at same site) and the larger circles denote the results of test 2 (i.e., teams at different sites).

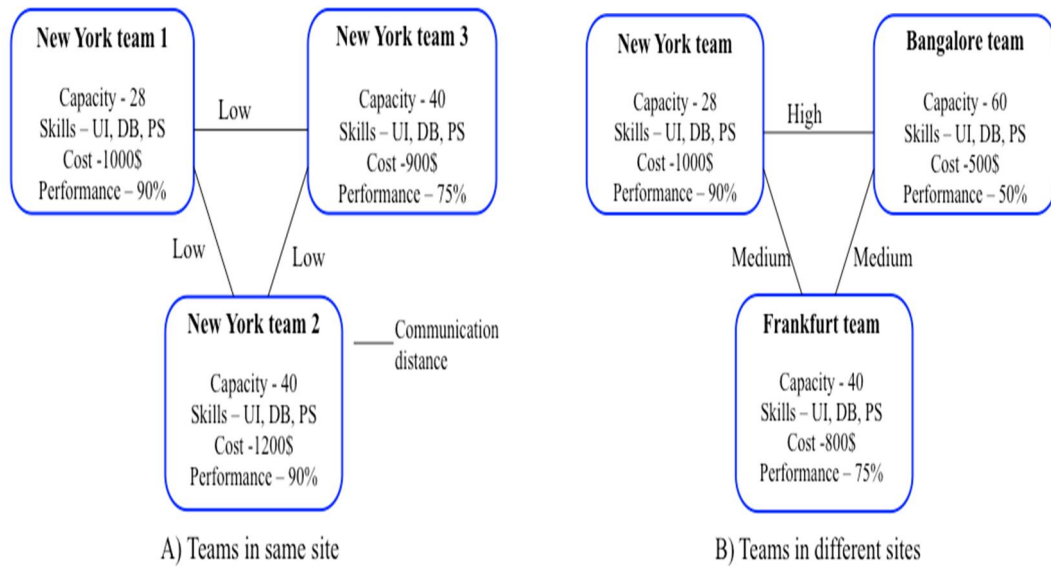


Figure 30. Resources used for test 1 and test 2

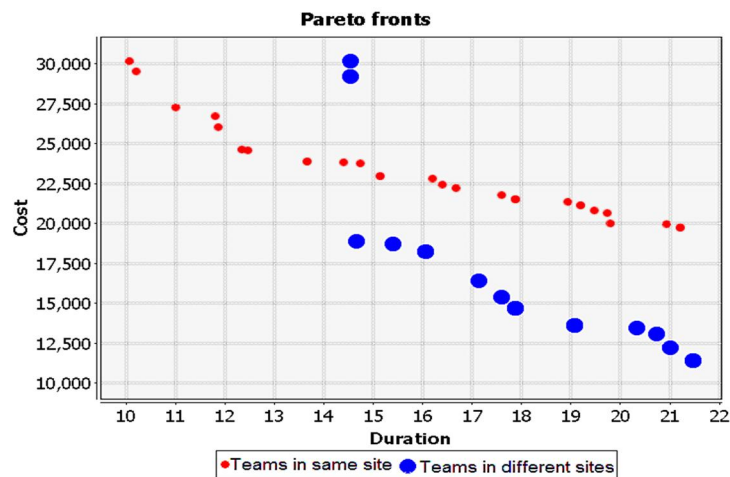


Figure 31. Non-dominated solutions of test 1 and test 2

As can be seen from Figure 31, duration is longer when work is distributed to remote teams. It is taking approximately 50% additional time to finish the project when work is distributed to remote teams, compared to teams at the same site. However, the cost to employ teams at the same site is higher than the cost to employ remote teams, as the teams at the selected site have high daily wages. The main reason for longer duration in

remote teams compared to same site teams is communication overhead. The communication overhead among local teams is less, as it is easier to communicate among local teams than remote teams.

To understand what factors influenced the duration of solutions in test 2, the work distributions corresponding to test 2 are analyzed. A work distribution proposal of test 2 is shown in Figure 32. The proposal shows the components allocated to teams and the order in which the team develops the components. As can be seen from the work distribution, to reduce the inter-team communication, the genetic algorithm has moved components with inter-dependencies to the same team. For example, in Figure 32, the inter-dependent components CoffeeMachine \rightarrow WaterControl and TemperatureRegulation \rightarrow HeaterDriver and DrapeDriver \rightarrow DrapeRegulation are assigned to same teams. In addition, the components are scheduled according to their precedence constraints, i.e., teams do not need to wait until preceded components are developed. However, the work allocation has resulted in inter-team communication between New York and Bangalore teams, New York and Frankfurt teams and Bangalore and Frankfurt teams. As these teams have medium to high distance between each other, the communication effort required between teams is much higher. Analysis of the work distribution confirms that the resulted communication overhead has hindered the development speed of work distributions in test 2.

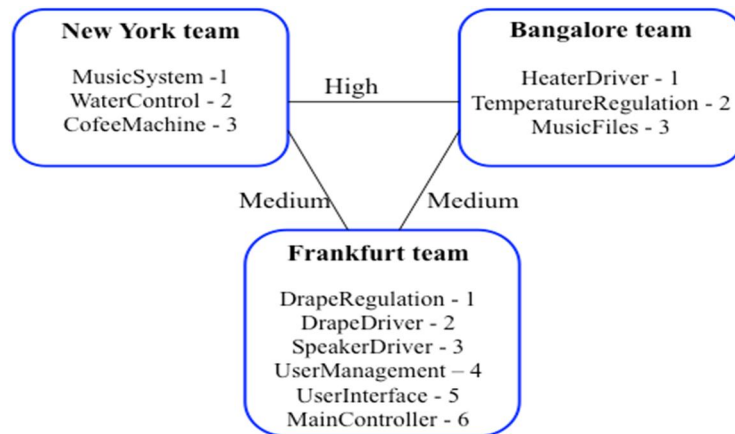


Figure 32. A work distribution result of test 2

To study question two, we have considered a situation where the development team at customer site (New York team) is not able to develop ehome project within agreed duration and cost. In this situation, the project manager needs to find additional teams to perform the work within agreed duration and cost. In addition to New York team, if the project manager is asked to choose either a team at Bangalore site (shown in Figure 33) or a team at London site to finish the work, then which team the project manager has to choose is a tradeoff between developing the project quickly and developing the project cheaply.

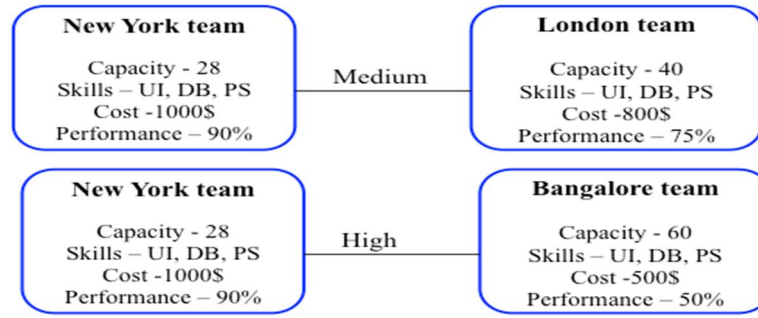


Figure 33. Characteristics of New York and London teams, New York and Bangalore teams

It is not straightforward to understand which site is better, as both sites have advantages and disadvantages with respect to each other. For understanding which site is better, two different tests are performed using the team models shown in Figure 33. The non-dominated solutions of the Pareto fronts of two tests are shown in Figure 34. The smaller circles denote results of test 1 (i.e., teams in New York-London sites) and the larger circles denote the results of test 2 (i.e., teams in New York-Bangalore sites).

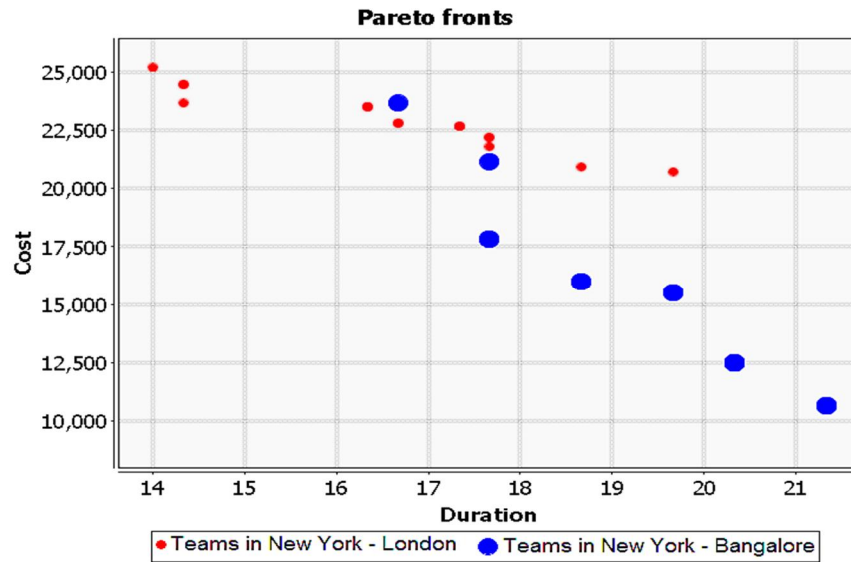


Figure 34. Non-dominated solutions of test 1 and test 2

As can be noticed from Figure 34, the project can be developed quickly if London team is used and the project can be developed cheaply if Bangalore team is used. However, with Bangalore team the duration required to complete the project is much longer than the London team. As can be seen, no team is better than other team with respect to both objectives. In this situation, the project manager has to choose a team based on the project completion time and cost available for developing the project. The project manager can use the Pareto fronts of tests to analyze how different work distributions influence the cost and duration objectives and can choose the suitable work distribution for a given project.

4.4 Summary

This chapter presented a method to use genetic algorithms for planning GSD projects. The genetic algorithm starts with a population of random initial work allocation proposals and optimizes the population over the generations with respect to cost and

duration objectives. The genetic algorithm has generated sensible work distributions and schedules and applied appropriate decoupling solutions with respect to the distant teams. Furthermore, this chapter presented the application of genetic algorithms for finding the approximations of Pareto optimal front for multi-objective project planning problem. As discussed in Subsection 4.3.2, the Pareto optimal genetic algorithm can be applied to study different project planning questions in the context of GSD. The estimates provided by the Pareto optimal genetic algorithm seem sensible, i.e., communication is major hurdle in global software development and hinders the development time. Also, the results are inline with the general assumptions about the cost and project completion time, i.e., shorter project completion time can be achieved at the expense of cost and vice-versa. This shows that the work distribution meta-model reflects the reality of the project scheduling problem.

Work allocation in a GSD project is a challenging task. The project manager has to evaluate a wide range of work allocation plans and schedule plans for coming up with a good work allocation plan. It is sensible to look for an automated approach to perform such a challenging task. The approaches presented in this chapter demonstrated the capability of genetic algorithms for solving such a challenging problem. We anticipate that if the approaches are extended to include the best practices related to the GSD, then they can act as a valuable support for project managers in distributing the work to distributed teams. Moreover, we noticed that the proposed approaches could be applied to solve new research problems as well, which are discussed in Section 10.2. At this stage, we expect that the work allocation estimations provided by the approaches can be used as recommendations, which can be further enhanced by the project manager based on her experience.

5 ALGORITHMIC APECTS OF SOFTWARE ARCHITECTURE SYNTHESIS

In this work, in addition to applying genetic algorithms for developing automated support for software architecture design and for planning GSD projects, some research is conducted on the applied algorithms. This chapter summarizes these investigations that are reported in publications [P2] and [P4]. The contents of this chapter correspond to the thesis contribution 6, i.e., “multi-objective genetic architecture synthesis using quality farms” and contribution 2, i.e., “software architecture design using constraint satisfaction and optimization”. This chapter is organized as follows. First, Section 5.1 discusses the quality farm approach for accelerating the genetic architecture synthesis in finding optimal solutions. Section 5.2 presents the application of constraint satisfaction techniques for designing the software architecture of a system. Finally, the chapter is summarized in Section 5.3.

5.1 Using Quality-farms in Genetic Architecture Synthesis

Software architecture design is also a multi-objective problem like software project planning. In the case of multiple quality objectives (e.g. modifiability, efficiency), the genetic architecture synthesis (described in Section 3.1) has a drawback that optimization of one objective may be achieved at the expense of other objective. One approach to handle multi-objectives is to use Pareto optimality (as discussed in Section 2.2) in the genetic architecture synthesis. The use of Pareto optimality in genetic architecture synthesis has been already studied in fellow researcher’s work [Räihä et al., 2011] and is also applied to planning GSD projects (see Section 4.3). However, the Pareto approach does not support intelligent merging of superior sub-solutions for different objectives and eventually relies on a human decision maker for choosing the solution, which is not always possible. Also, as the approach starts with a random initial population the number of generations required to reach the optimal solutions may be higher. Thus, an alternative approach is needed to solve the multi-objective problem in genetic architecture synthesis and to accelerate the progress of genetic architecture synthesis in finding optimal solutions.

When applying the genetic algorithm to solve a given problem, it is necessary to refine on all the major components of genetic algorithm, such as population initialization, mutation and crossover operators, fitness assignment, selection operators, and so on, in order to find a good solution to the problem at hand [Gen, 2008]. In this work, a new population initialization method for genetic algorithms is studied for accelerating and optimizing the multi-objective problems. The method uses farms, i.e., separate populations that are evolved independently under the emphasis of single objectives for creating the initial population.

5.1.1 Method

The idea is that each quality objective has its own farm, i.e., a separate population evolving under the emphasis of a single objective. After certain number of generations, the farms are crossbred: parents are selected from different farms so as to combine superior solutions of separate objectives, developed independently in the farms. The resulted new population from crossbreeding is used as a starting population for the genetic algorithm to explore the possibilities for more fine-grained improvements.

Figure 35 presents the use of quality farms in genetic architecture synthesis for optimizing two quality objectives, i.e., efficiency and modifiability. In the farming stage (shown in Figure 35), two farms are created one for modifiability and one for efficiency. The individuals in the efficiency farm are evolved under a fitness function which emphasizes efficiency of the system. Similarly, the individuals in the modifiability farm are evolved under a fitness function that emphasizes modifiability of the system.

After the farming stage, we have two farms, where each farm has good individuals with respect to one objective. The elites of two farms are then crossbred to create a new population. As the crossover point in a single-point crossover is chosen randomly, there is a chance that best parts of individual that have big impact on the quality requirements may be lost. Thus, to consider the strengths of individuals from both farms in crossbreeding, a *complementary gene-selective* crossover [Räihä et al., 2010] is used. It inspects the individuals in detail and purposefully combines the best parts (i.e., the parts of the individuals that have biggest impact in satisfying the quality objectives) of both the parents and produces one offspring. Here, best individuals from efficiency and modifiability farms are subjected to complementary gene-selective crossover to form the new population, which contains offspring that are good in both modifiability and efficiency. In addition, those individuals in the elite not taking part in the crossover are inserted to the new population. Finally, the newly formed population is given as starting point for the genetic algorithm and is executed with a fitness function that rewards balanced modifiability and efficiency.

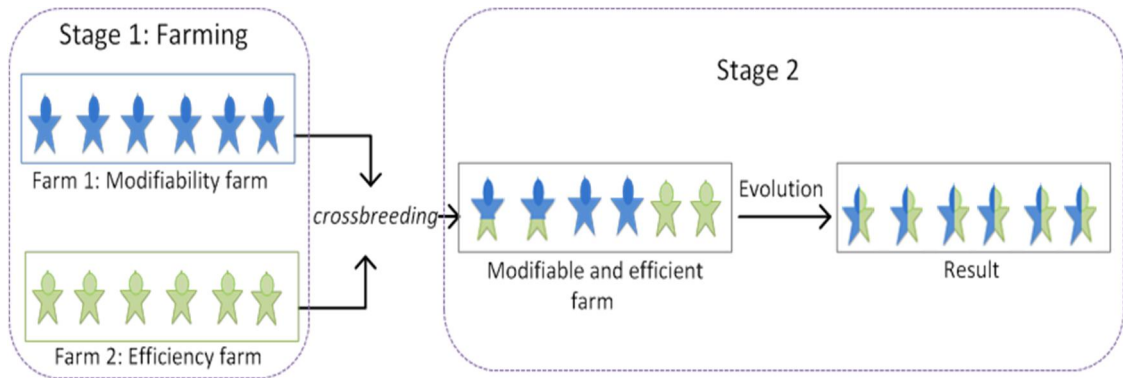


Figure 35. Farm approach for optimizing two quality objectives

5.1.2 Experiments and Evaluation

The goal of the experiment was to study the effectiveness of the quality farm approach in accelerating the genetic architecture synthesis towards architecture proposals that are good in both modifiability and efficiency. The experiment reuses ehome system given in Appendix B and described in chapter 4. In this experiment, the fitness function described in Subsection 3.1.4 is used. As the objectives to be optimized here are modifiability and

efficiency, the complexity sub-fitness function is not taken into account in calculating the fitness values. Also, the mutation probabilities that are found after some experimentation are used in the experiment. The fitness graphs reported in the experiment are average values of 20 runs.

Firstly, the modifiability and efficiency farms that utilize a population of 100 individuals were evolved for 100 generations. Modifiability sub-fitness is weighted over other sub-fitness functions in the modifiability farm and efficiency sub-fitness is weighted over other sub-fitness functions in the efficiency farm. The other quality objective is kept in the farm so that the individuals in each farm also offer some solutions to the other objective. The corresponding overall fitness graphs (including modifiability and efficiency sub-fitness values) are shown in Figure 36. As the patterns used in the genetic architecture synthesis are generally aimed to improve the modifiability of the system, the overall fitness of the modifiability farm improved as the evolution progressed. However, this phenomenon is not seen in the efficiency farm. This is because the architecture is very efficient in the initial design phase, where there are no indirect connections or patterns. As the population is evolved through mutations and crossover, the number of connections between classes increases, which degrade the efficiency.

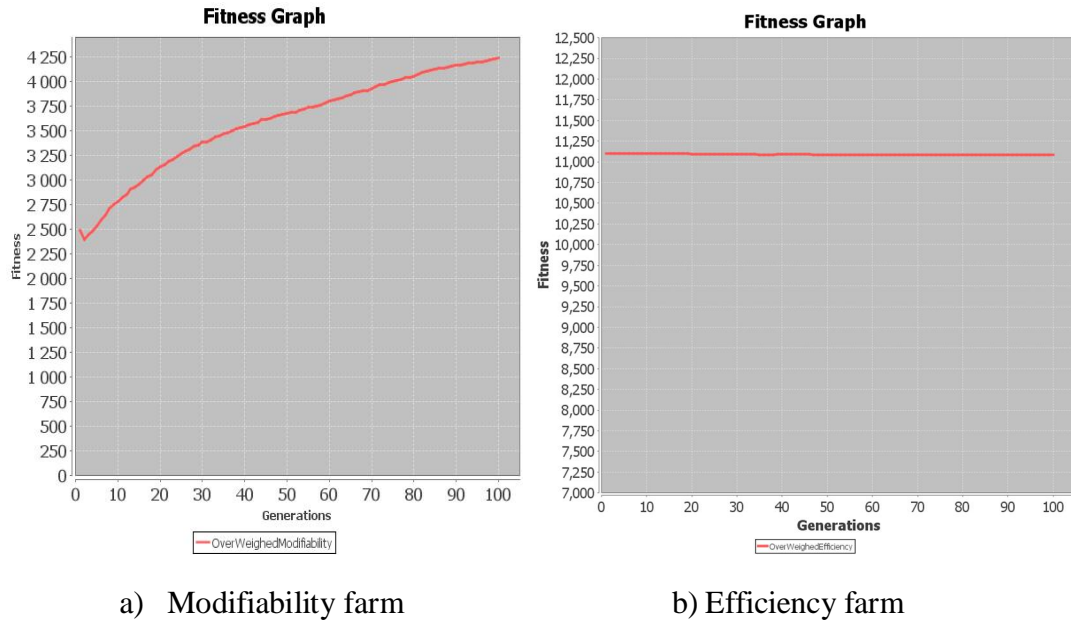


Figure 36. Modifiability and Efficiency farms

The modifiability farm and efficiency farm are crossbred in the second stage. The population resulted from crossbreeding is used as initial population for genetic algorithm and is executed for 500 generations. The fitness function is weighted to reward modifiability and efficiency sub-fitnesses equally. The overall fitness graph resulted after evolving the crossbreed population is shown in Figure 37, where the first 200 generations are used for evolving the independent quality farms. The efficiency and modifiability sub-fitness graphs of farm stage are shown in Figure 38, where the thin curve represents efficiency sub-fitness and the thick curve represents modifiability sub-fitness. As can be seen from Figure 38, the difference in the efficiency and modifiability fitness values of the individuals is decreasing over the generations. This shows that as evolution progresses, the genetic algorithm tries to achieve a compromise between modifiability and efficiency.

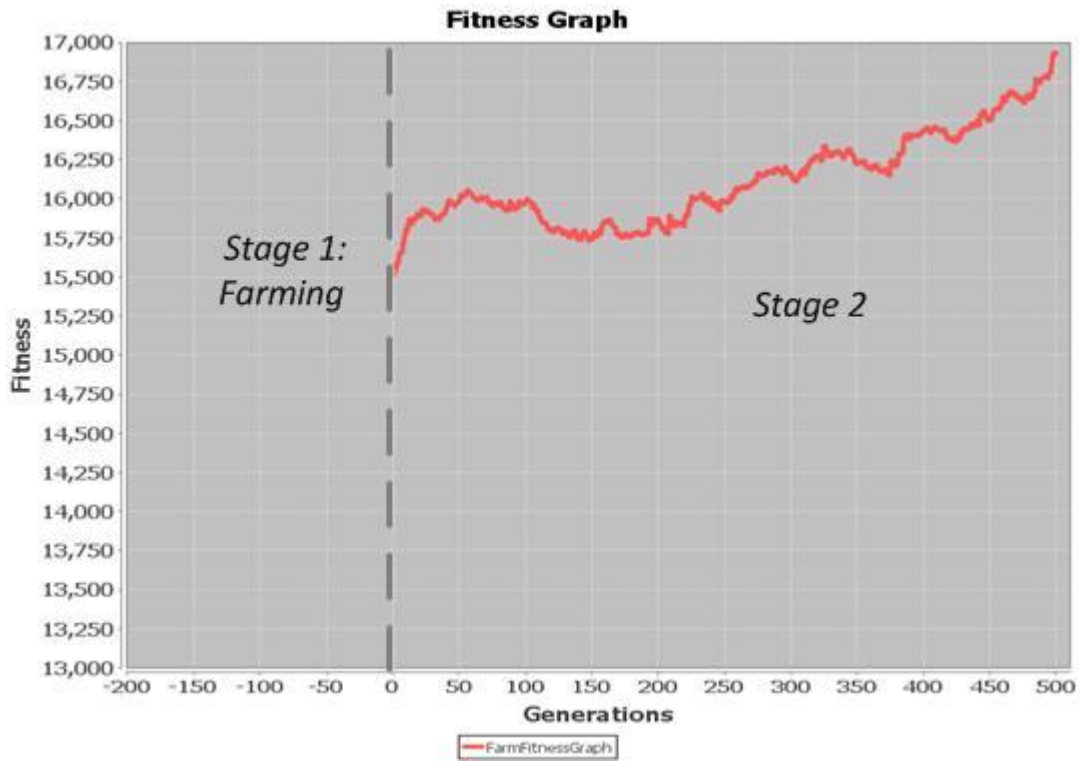


Figure 37. Fitness graph of second stage genetic algorithm (i.e., farm approach)

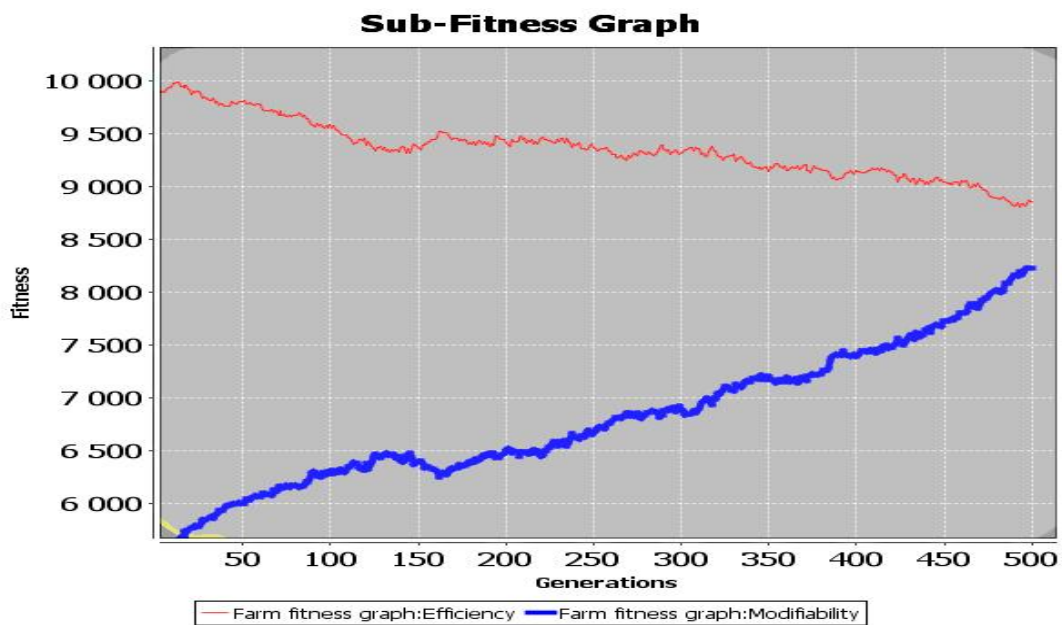


Figure 38. Sub-fitness graphs of stage two (i.e., farm approach)

Secondly, it was tested whether the use of quality farms accelerated the progress of genetic architecture synthesis towards optimal solutions. This was tested by comparing the results of the farm approach with the results of normal genetic architecture synthesis (starting with a random initial population). The parameters used for the basic approach are the same as the parameters in stage 2 of the farm approach. The fitness graph of normal genetic architecture synthesis is presented in Figure 39.

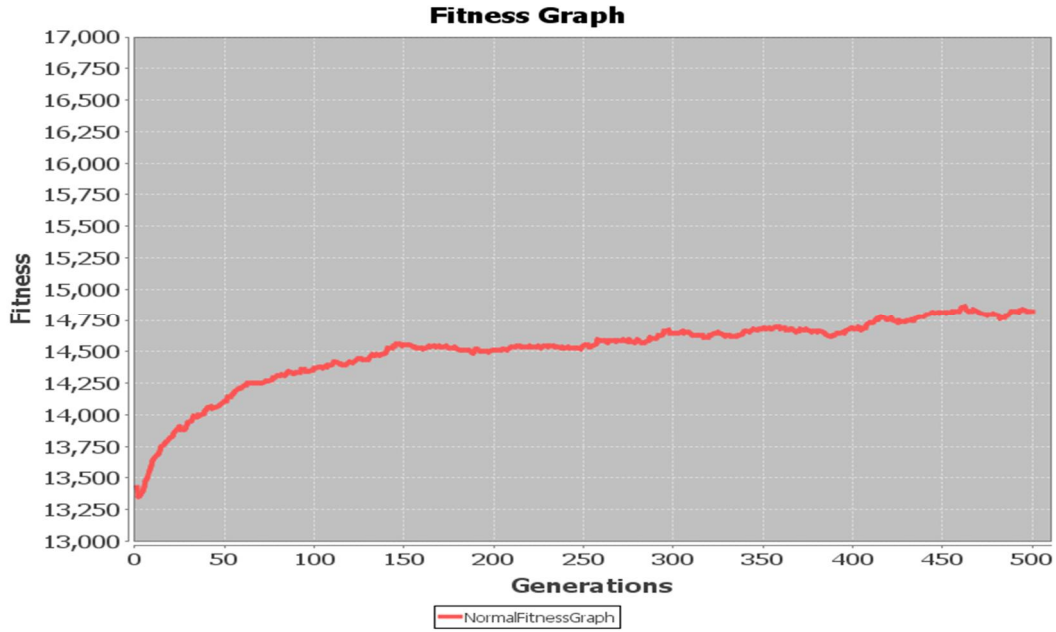


Figure 39. Fitness graph of normal genetic architecture synthesis

As can be seen, compared to the genetic architecture synthesis approach shown in Figure 39, the fitness achieved with the farm approach shown in Figure 37 is significantly higher. At the point of 300 generations in Figure 37, the second stage genetic algorithm has had totally the same time to work with the population as with the normal genetic architecture synthesis at the end of the 500 generation long evolution. The difference between the end value of the basic evolution (approximately 14750) and the farm approach after the same evolutionary time (approximately 16250) is about 1500 units. This indicates that given the same amount of time, the farm approach can produce better architecture proposals. Moreover, even after 500 generations the normal genetic architecture synthesis was not able to achieve the starting fitness of the farm approach. This shows that initial population has significant effect on the genetic algorithm in finding optimal solutions.

Overall, the fitness curves show that given the same amount of time, the farm approach can produce better architecture proposals significantly faster than the genetic architecture synthesis with random initial population. Also, the results are in line with existing studies [Burke et al., 2004; Zitzler et al., 2000], emphasizing the influence of the initial population in the genetic algorithm.

5.2 Using Constraint Satisfaction Techniques for Software Design

Software architecture design is a decision making process. During the architecture design process, a software architect makes multiple decisions related to the system at hand. However, in many cases, the rationale behind the chosen decisions is not documented [van der Ven et al., 2006]. Eventually, the rationale behind the selected decisions vaporizes. This rationale is useful in some cases, such as for reusing design decisions, managing architectural knowledge in the organization [de Boer and Farenhorst, 2008] and for decision centric architecture design [Jansen, 2008]. The genetic architecture synthesis approach described in Chapter 3 is able to make sensible design decisions, but due to the stochastic nature of the genetic algorithm, it is difficult to produce the rationale behind the chosen decisions. Moreover, expressing new patterns

and introducing new quality attributes for genetic architecture synthesis requires considerable amount of work, as the patterns and the fitness function are hard-coded. Therefore, the application of constraint satisfaction techniques has been studied as an alternative approach for software architecture synthesis.

As in Section 3.1, software architecture is designed by applying a set of architecture design decisions (i.e., patterns) to the initial design of the system. In order to use constraint satisfaction techniques for designing the software architecture of a system, the patterns are expressed in terms of role models. Many studies have specified design patterns as role models [Riehle, 2000; A.L. Guennec et al., 2000; Kim et al., 2003]. An object can participate in several *roles* in the object collaboration [Riehle, 2000]. A role is further associated with a *type*, which specifies the type of elements that can participate in that role. For example, a role can be of type *class*, which specifies that only class elements can take that role. Moreover, in order to apply a pattern, a role may require that the same element within the design pattern never plays another role or two roles might mutually require each other, etc. These requirements specify rules for binding software elements to roles. This work also uses similar role models for expressing the patterns like design patterns or architecture styles (discussed in Section 2.5).

In this work, the main focus is on finding the optimal assignment of patterns in the initial design, represented by the bindings. For example, consider the Adapter design pattern discussed in Section 2.5. The Adapter pattern converts the interface of a class to another interface and allows classes to work together, which would not be possible before because of incompatible interface. The role types in the Adapter pattern can be *class* and *operation*. The class roles in the Adapter pattern are the adaptee class (denoted as Adapter supplier class) and the client class (denoted as Adapter client class). The operation roles in the Adapter pattern are the operation in the Adapter client class that needs the service, and the service operation in the Adapter supplier class. The corresponding operation roles are denoted as Adapter client operation and Adapter supplier operation. When Adapter pattern is instantiated, these are the roles that must be bound to the software elements. Moreover, the relationships between the roles must present between the software elements bound to the roles.

The Adapter pattern defines certain relationships between the roles. For example, consider the Adapter client operation role. The Adapter client operation role has dependency relationship with Adapter supplier operation role and containment relationship with Adapter client class role. From this relationship, the rules for binding Adapter client operation role can be stated as: (i) the element bound to the role must be an operation, (ii) this operation must have a dependency relationship with an operation bound to Adapter supplier operation role, and (iii) the class of Adapter client operation has to be bound to Adapter client class role. Similarly, the rules for binding all the roles in the Adapter pattern can be derived.

An Adapter pattern can be applied if all the roles of the Adapter pattern are bound to some elements of initial design and the requirements between the roles hold between the bound elements. To illustrate this, let us consider the application of Adapter pattern to a fragment of initial design of ehome shown in Figure 40. This fragment contains coffee machine subsystem which controls coffee making process. As there can be multiple ways in connecting WaterControl to CoffeeMachine, it is assumed that in the future WaterControl component may be replaced by other third party component that provides the same logical services. To make the rest of the system independent of the interface of

the WaterControl component, the Adapter pattern can be applied. Figure 40 shows the binding of roles of the Adapter pattern to Coffee machine subsystem of ehome. For presentation purpose only few methods of CoffeeMachine class are shown in the picture.

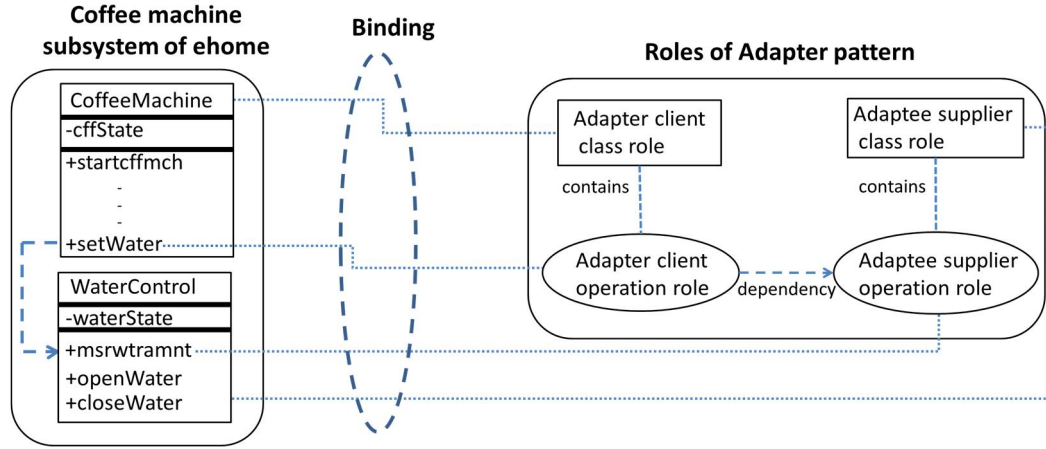


Figure 40. Overview of applying Adapter pattern to coffee machine subsystem of ehome

As can be seen from Figure 40, the elements of the initial design are bound to the roles of Adapter pattern and also the requirements between the roles are hold between the elements of the initial design. For example, Adapter client operation role is bound to setWater operation and Adaptee supplier operation role is bound to measure water amount (msrwtramnt in Figure 40) operation. Moreover, the relationship (i.e., dependency) between the roles is also present between these operations.

5.2.1 Constraint Satisfaction and Optimization Approach to Software Design

In this work, the problem of binding patterns to the initial design of the system (as presented in Figure 40) is solved by modeling it as a CSOP. As discussed in Section 2.3, a CSOP consists of a set of variables, a set of constraints and objective function. Each variable is associated with a set of possible values, known as its domain. A constraint is a relation defined on some subset of these variables and denotes valid combinations of their values. The goal of a CSOP is to find assignments of values to variables that satisfy all the constraints and either maximize or minimize the objective function.

The mapping of CSOP elements to software architecture design is shown in Table 6. The elements of initial design (e.g. classes, attributes and operations) are mapped to *variables*. Every element in the initial design that can participate in a pattern will be a variable. The type of the element, i.e., class, attribute or operation will become the type of the variable. For example, in Figure 40, the variables of type class are CoffeeMachine and WaterControl and the variables of type operation are setWater, openWater, etc. This process of obtaining variables from initial design is termed as *variable extraction*.

The *values* of the variables are obtained from the roles of the patterns. As specified earlier, a pattern role can be associated with a type. For each variable, all the roles that have a compatible type are the possible values. For example, in Figure 40, value domain for class variable CoffeeMachine consists of Adaptee supplier class and Adapter client class roles and value domain for operation variable setWater consists of Adapter client operation and Adaptee supplier operation roles. This process of obtaining values from patterns is termed as *value extraction*.

Constraints restrict the association of values to variables. Constraints are used to find all the designs that are valid. There are two types of constraints, which are *pattern constraints* and *problem constraints*. The pattern constraints are gathered from the requirements to be hold for applying a pattern. For example, in Figure 40, a pattern constraint for the Adapter pattern states that if the value of setWater is Adapter client role and the value of measure water amount (i.e., msrwtramnt in Figure 40) is Adaptee supplier role, then operation setWater must have dependency relationship with operation measure water amount. This constraint ensures that the corresponding precondition is satisfied. The problem constraints are additional constraints that can be given by the architect depending on the requirements of the system to be built. For example, problem constraints could restrict or forbid the application of certain patterns. An example of a problem constraint is presented in the next Subsection.

Finally, an *objective function* is used to find an optimal association of patterns to initial design from all the valid associations. The objective function can be modeled to evaluate the quality of the architecture based on how well pattern roles are applied to the elements of initial design. The aim is to find an architecture that either maximizes or minimizes the value of the objective function.

Table 6. Modeling software architecture design as a constraint satisfaction and optimization problem

<i>Elements of constraint satisfaction and optimization problem</i>	<i>Software architecture design</i>
Variables	Classes, operations and attributes of initial design
Values	Roles of the available patterns
Constraints	Rules for binding roles of patterns to elements of initial design
Objective function	Evaluation criteria for measuring the quality of the architecture

5.2.2 Experiment and Evaluation

The goal of the experiment was to study the suitability of constraint satisfaction and optimization approach for designing software architectures. The experiment reuses ehome system given in Appendix B and described in Chapter 4. The chosen patterns for designing ehome system are a high-level architectural style (Message Dispatcher), structural design patterns (Adapter and Façade) and behavioral design patterns (Strategy and Template Method). The roles of the chosen patterns are obtained in the same way as described for the Adapter design pattern. A constraint programming system PaLM (Propagation and Learning with Move) [Jussien and Barichard, 2000] has been used for implementing the CSOP representation of ehome. The PaLM system has the capability to explain questions, such as why is a variable assigned to a value, why the problem does not have any solution, etc.

The first step in applying the CSOP approach to design the architecture of ehome is to extract variables from the initial design of ehome. The variables are extracted following the variable extraction process described earlier. As the chosen patterns deal with two kinds of elements, classes and operations, the variables are classified into two types, i.e., class and operation variables. The class variables include all classes of ehome and the operation variables include methods of classes. The values for variables of type class are all the pattern roles of type class and the values for operation variables consist of all the pattern roles of type operation. The value extraction process described earlier is used for extracting the values. Moreover, to denote the absence of a pattern, class variables are associated with a special role, which is “No pattern class” role. Similarly, operation variables have one special role, which is “No pattern operation” role. The pattern constraints are obtained directly from the preconditions of the pattern in the same way as described earlier for the Adapter pattern. The total number of pattern constraints used in the experiment is 38. Typically, two to three constraints are specified for each pattern role. The problem constraints may concern the sensible application of a pattern in a particular domain or system. For ehome system, a problem constraint states that at most five components are allowed to use messaging in their communication.

The objective function has been applied to evaluate how well patterns are introduced into the initial design of the target system. In this experiment, an abstract objective function has been used to evaluate the modifiability and efficiency of the architecture. A more fine-grained fitness function considering different quality attributes is needed for real world situations. The objective function $f(x)$ for a solution x is given in equation (10).

$$f(x) = w_1 * sf_1 + w_2 * sf_2 \quad (10)$$

where w_i is the weight for the respective sub-fitness sf_i and sf_1 measures modifiability and sf_2 measures efficiency. The goal is to maximize the objective function.

The modifiability sub-function rewards the design, taking into account how well patterns are applied to satisfy the expected variation of operations. In computing the reward, the variability of operations is taken into account. In addition, the modifiability sub-function also penalizes the direct calls, which creates strong coupling among the operations and their hosting components. The efficiency sub-function rewards classes that are not communicating through patterns. It also penalizes the calls through patterns, as the calls through patterns are slow. The frequency of operations is used for emphasizing the penalty. Compared to fitness function given in equation (3), the objective function used here is not separated to consider the positive and negative effect of coupling and cohesion metrics on the modifiability and efficiency of the system. Also, the effect of Client/Server pattern, the use of interfaces and the complexity of the system is not considered in evaluating the quality of the design.

The best architecture of ehome system obtained from the experiment is shown in Figure 41. As can be seen, the solution uses Message Dispatcher, Adapter pattern and a couple of Template Methods and Strategy patterns. The major subsystems are communicating via messaging, which is a natural solution for this kind of system. Moreover, as modifiability is emphasized in the objective function, the introduction of these patterns is sensible.

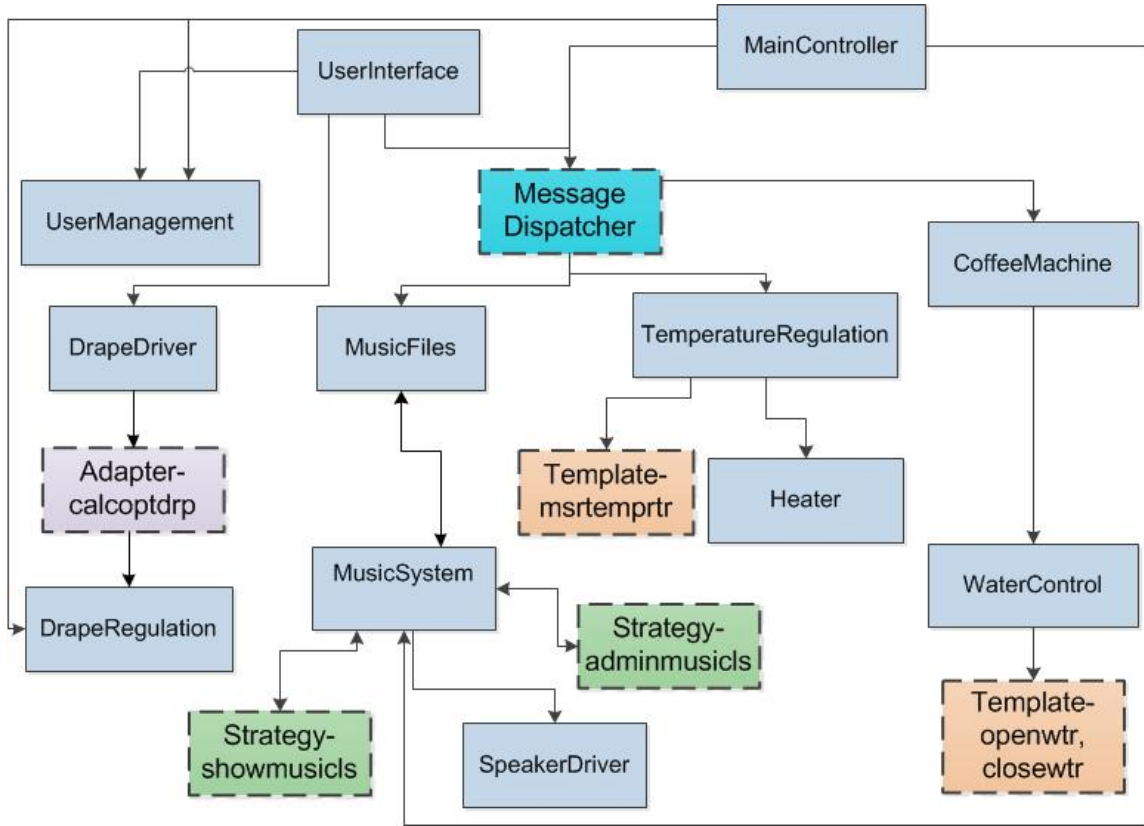


Figure 41. Proposal for ehome system

In the design shown in Figure 41, the MusicSystem class does not take any pattern role. For example, it can take the role of Dispatcher client class, as it has dependency with MusicFiles class, which has the role of Dispatcher supplier class. To understand why such a decision is made the solver is queried for “Why MusicSystem class variable does not take the role of Dispatcher client class”. The solver provided the following explanation:

“Why MusicSystem \neq Dispatcher client class : {message usage constraint = number of components using messaging : 5 {UserInterface, MainController, MusicFiles, CoffeeMachine, TemperatureRegulation}}”

The cryptic expression of the solver can be opened as follows: the constraint specified on the capacity of the Message Dispatcher usage limits the MusicSystem class variable from taking the role of Dispatcher client class. The constraint states that at most five components are allowed to communicate using messages. In addition, objective function is one other reason for this result. As the proposal with the current set of allocated variables to values is the best proposal, it means that other allocation combinations, where MusicSystem class has Dispatcher client class role have low values.

The experiment shows the ability of constraint satisfaction and optimization approach in designing the software architecture of a system. The approach also provides the capability to reason about the chosen design decisions. The explanations provided are cryptic and bit difficult to interpret, but they still give an initial idea about why a certain decision is made. Moreover, at this stage, the approach is only seen as a recommendation system that can support the human architect in designing high quality software systems.

When compared to genetic architecture synthesis described in Section 3.1, the constraint satisfaction and optimization approach has the capability to produce the rationale behind the chosen solution, as a functionality of the constraint solver. Also, the modeling language provided by the solver can be used for easily extending the approach with new patterns for designing the architecture of a target system. However, in case of large-scale systems, the space of possible architecture proposals is far too large to use the constraint satisfaction and optimization approach for finding the optimal architecture proposals in a reasonable time. Moreover, a further study needs to be performed to evaluate the performance of constraint satisfaction based architecture design and genetic architecture synthesis approach in finding optimal architecture proposals.

5.3 Summary

This chapter discusses the idea of using quality farms in genetic architecture synthesis. The approach uses farms, i.e., separate populations that are evolved under the emphasis of a single objective. After certain generations, the farms are crossbred to form a single population, which is then subjected to a normal genetic architecture synthesis process. Furthermore, this chapter also presented the application of constraint satisfaction methods to the software architecture design problem. The approach demonstrated the capability of constraint satisfaction methods for producing the rationale behind the chosen design solution, which could be very valuable in the software architecture design process.

In the initial experiments conducted with the quality farms, we observed that the use of quality farms has accelerated the genetic architecture synthesis in finding the architecture proposals. However, further research needs to be done to evaluate the applicability of the approach in case of more than two objectives. Understanding rationale behind the chosen architecture solutions is very important in software architecture design. It enables the reuse of design experience. The proposed constraint satisfaction and optimization approach to software architecture design can be extended towards a tool for understanding the rationale behind the applied architecture decisions. Moreover, we anticipate that by introducing a growing collection of patterns and tacit knowledge of architects, the tool can be further developed for managing software architecture knowledge.

PART III – TOOL SUPPORT FOR SOFTWARE DESIGN AND PROJECT PLANNING

This part describes the tool support developed for software architecture design and project planning. In addition, techniques to collaborate human decision maker with the automated support are also presented.

6 TOOL SUPPORT

This chapter presents the Darwin tool developed for supporting the genetic algorithm-based architecture design and project planning approaches described in Chapters 3 and 4. It summarizes the tool support presented in publications [P1, P6]. The tool has been first described in publication [P1] to support software architecture design and has been further extended in publication [P6] to support project planning. The tool was initially developed to support the research team, but later a goal has been to experiment features that might support practitioners. The contents of this chapter correspond to the thesis contribution 4, i.e., “tool for software design and project planning”. This chapter is organized as follows. First, Section 6.1 gives a brief introduction to the user interface provided by Darwin. The architecture of Darwin is presented in Section 6.2. The usage of Darwin is briefly discussed in Section 6.3. Section 6.4 summarizes the Darwin tool.

6.1 User Interface of Darwin

Darwin provides user interface for giving the initial design and quality requirements of the system, for expressing the teams available in the organization and for setting the parameters of the genetic algorithm. Moreover, it provides a way for visualizing and studying the results, i.e., fitness graphs, resulted architecture proposals and work distribution proposals produced by the genetic algorithm.

The user interface of Darwin consists of several views, as shown in Figure 42. The *evolution view* can be used to create, save, open and remove an *evolution*, which is the data model that holds information about one genetic evolution. It stores the input given by the user (e.g. initial design, genetic parameters) as well as the outputs (e.g. fitness graphs, architecture proposals) produced by the genetic algorithm.

As discussed in Section 3.2, the input for genetic architecture synthesis consists of initial design of the target system, quality requirements and parameters of the genetic algorithm. The *use case*, *sequence and class diagram views* of Darwin can be used to create the initial design of the target system. The *weights view* can be used for adjusting the fitness weights of different sub-fitness functions. The parameters of the genetic algorithm can be given using the *mutations* and *settings view*. The mutations view contains user interface controls for adjusting the mutation probabilities as well as crossover probability. The population size and number of generations to be used in the genetic algorithm can be adjusted in the *settings view*. Furthermore, the mutations view, weights view and settings view are associated with default values that can be used in applying the genetic algorithm for software architecture design. These values are obtained after some experimentation.

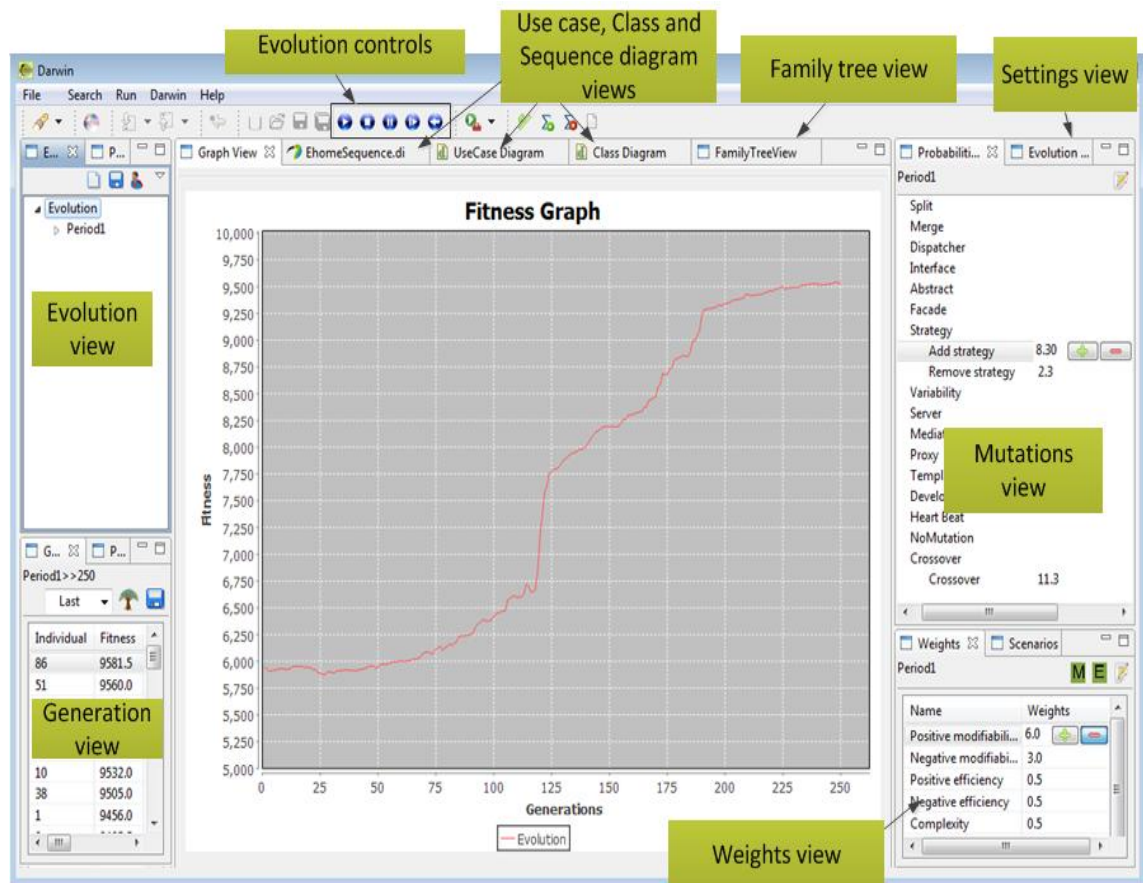


Figure 42. Darwin user interface

An evolution can be executed using the evolution controls (see the upper part of Figure 42). The evolution controls can be used for starting, pausing, and resuming an evolution. While genetic algorithm processes the population of chromosomes, the fitness of the generation is visualized in the *fitness graph* view. The fitness can be viewed as either fitness of the best individual in a generation or average fitness of the elite individuals. The settings view can be used to change the way the fitness graph displays the calculated fitness values. On the fitness graph, y-axis represents the fitness (elite average or best individual) and the x-axis represents the generation number. The resulted individuals, i.e., architecture proposals can be examined in the *generation* view. Darwin visualizes the architecture of an individual in the form of a class diagram in the class diagram view (the same as used for input). The individuals can be further examined using *family tree* view, which shows the family tree of the individuals involved in the evolution as a graph. The individuals and their parent information are shown in this view, allowing the exploration of the development of a particular family line.

To provide a tool support for genetic algorithm-based project planning (discussed in Section 4.2), a new version of Darwin has been made by extending the Darwin tool described earlier. When compared to the original Darwin (shown in Figure 42), the changes in this version are replacing the genetic algorithm for architecture synthesis with the genetic algorithm for project planning and adding a new view called *organization* view. The organization view (shown in Figure 43) is added for specifying the information related to teams available for developing the project. It can be used to specify team characteristics, view the existing team characteristic and modify the team characteristics. Moreover, in this version, the existing views of Darwin, such as

generations view, mutations view and weights view are modified to be suitable for the GSD problem context. For example, the mutations view is modified to include mutations for change team and change development order.

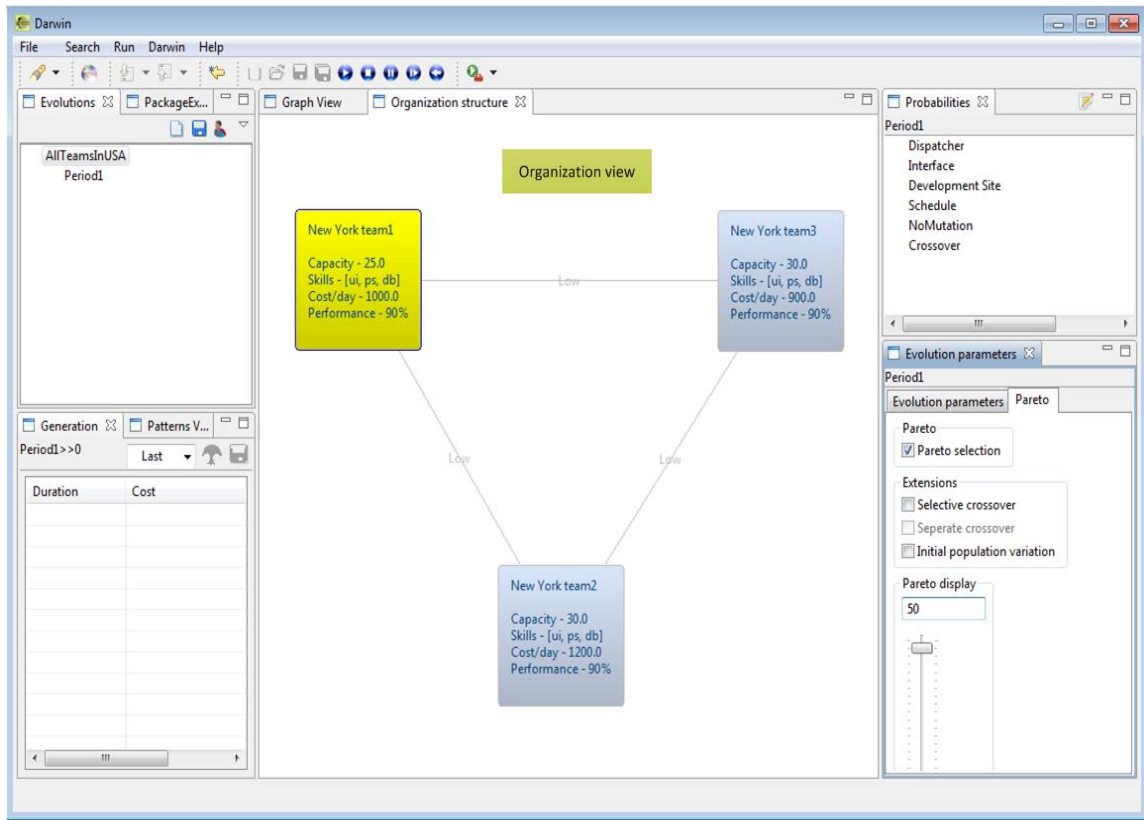


Figure 43. Darwin for planning projects

The input for genetic algorithm-based project planning (discussed in Section 4.2) consists of components and their corresponding operations and relationships, cost and duration sub-fitness weights, team resources and parameters of the genetic algorithm. The components information and parameters of the genetic algorithm can be specified using UML diagram views and mutations and setting views, as described above. The modified weights view can be used to specify the cost and duration sub-fitness weights. The organization view can be used for giving the information about the teams available for developing the project and modifying the team characteristics. After giving the input, the genetic algorithm for project planning can be executed using the evolution controls and the results can be viewed using the fitness graphs and generation view.

6.2 Architecture of Darwin

The architecture of Darwin follows Model-View-Controller (MVC) pattern [Krasner and Pope, 1988]. An evolution forms the model part of the pattern. As specified earlier, an evolution contains all the information including the inputs provided by the user as well as the outputs produced by the genetic algorithm. Different views of Darwin (e.g. evolution view, generations view and mutations view) discussed in the previous section form the View part of the pattern. The control logic for keeping the model consistent with the changes in the user interface resides in the views of Darwin.

Darwin makes use of the plugin-based architecture of Eclipse, which is a well-known integrated development environment [Eclipse, 2015]. The plugin architecture of Eclipse provides the facility to interact with other existing plugins of Eclipse. As shown in Figure 44, Darwin itself is a plugin which uses the user interface elements of Eclipse for providing its functionality. In the version for architecture design, the genetic algorithm engine plugin wraps the genetic algorithm described in Section 3.1 and in the version for project planning the genetic algorithm described in Section 4.2 is wrapped inside the genetic algorithm engine plugin. In addition, Darwin relies on several third-party plugins, such as UML2Tools [MDTtools, 2015], Papyrus [MDTtools, 2015], JFreeChart [JFreeChart, 2015] and Zest [Zest, 2015]. Darwin uses UML2Tools plugin for creating use case and class diagram views. UML2Tools is a CASE tool of Eclipse and provides editors for creating use case, class diagrams and sequence diagrams. However, the sequence diagram editor of UML2tools was not stable and was error-prone. Therefore, to create sequence diagrams Darwin relies on Papyrus tool, which is an advanced CASE tool of Eclipse. The JFreeChart plugin is used for creating the fitness graph view. Finally, to realize family tree view and organization view, Darwin uses the Eclipse’s visualization toolkit Zest.

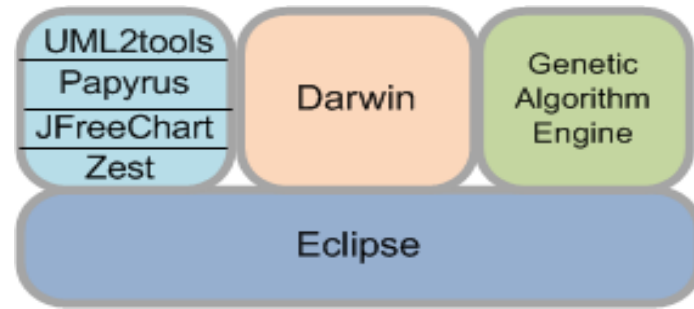


Figure 44. Darwin plugin

6.3 Use of Darwin

This section describes how to use Darwin for designing the software architecture of a system and for planning a project. Ehome system given in Appendix B is used as a target system. First, using the evolution view, an evolution for storing the information related to ehome has to be created. The second step is to express the initial design of ehome. The initial design can be incrementally created (as given in Section 3.2) using the use case and sequence diagram views. The created sequence diagrams can be then automatically transformed into initial design using controls provided in the evolution view of Darwin. Moreover, Darwin also provides support to express initial design as a text file.

After giving the functional requirements of the target system, the next step is to give the parameters of the genetic algorithm, which are population size, the number of generations, fitness weights and mutation and crossover probabilities. In this example, the default values provided by the settings view and mutations view are used for setting the parameters of the genetic algorithm. In the weights view, the modifiability sub-fitness is slightly outweighed over other sub-fitness functions. The fitness displayed on the fitness graph is set as average of 10 best individual’s fitness. After giving the parameters of the genetic algorithm, the next step is to start the simulated evolution using the evolution controls. As the evolution progresses, the fitness information is

portrayed on the fitness graph view, as shown in Figure 42. The increasing fitness values over the generations indicate that the genetic algorithm has gradually improved the quality of architecture proposals. In addition, the fitnesses of different sub-fitness graphs can also be viewed on the fitness graph. They can be used for exploring the effect of individual fitnesses (e.g. modifiability fitness, efficiency fitness) on the overall fitness. Also, the parameters of the genetic algorithm can be modified during the evolution and the effect of the modified values can be immediately observed in the produced results.

The individuals of each generation can be explored in the generations view as shown in Figure 42. It can be further used to view the architecture proposals and family tree views of the individuals in different generations. Exploring the family trees of the individuals can provide more information about the development of architecture proposals to the user. The best architecture proposal of the run is shown in Figure 45. As it is difficult to show the entire architecture proposal in one page, a fragment of best architecture proposal produced is shown. The genetic algorithm has introduced patterns, such as Message Dispatcher, Client-Server, Adapter and Template Method into the initial design. For the newly generated classes and interfaces names are created automatically (e.g. Adapter40, TemplateClass57). The names are derived according to the introduced pattern, with a unifying suffix.

In a similar fashion, Darwin version for project planning can be applied for obtaining work distribution proposals for a GSD project. The different components of ehome and their relationships can be expressed using the class diagram view. The team information can be expressed using the organization view (as shown in Figure 43). Again in this example, the default values provided by settings view, weights view and mutations view are used for setting the parameters of the genetic algorithm. The next steps are applying the genetic algorithm and viewing the results. These steps can be carried out in a similar manner as described above in the architecture design phase. A fragment of work distribution proposal resulted for ehome is presented in Figure 46. The proposal shows the components allocated to New York team1 and New York team3 and applied decoupling solutions between the components. The decoupling solutions are indicated with grey color in the figure.

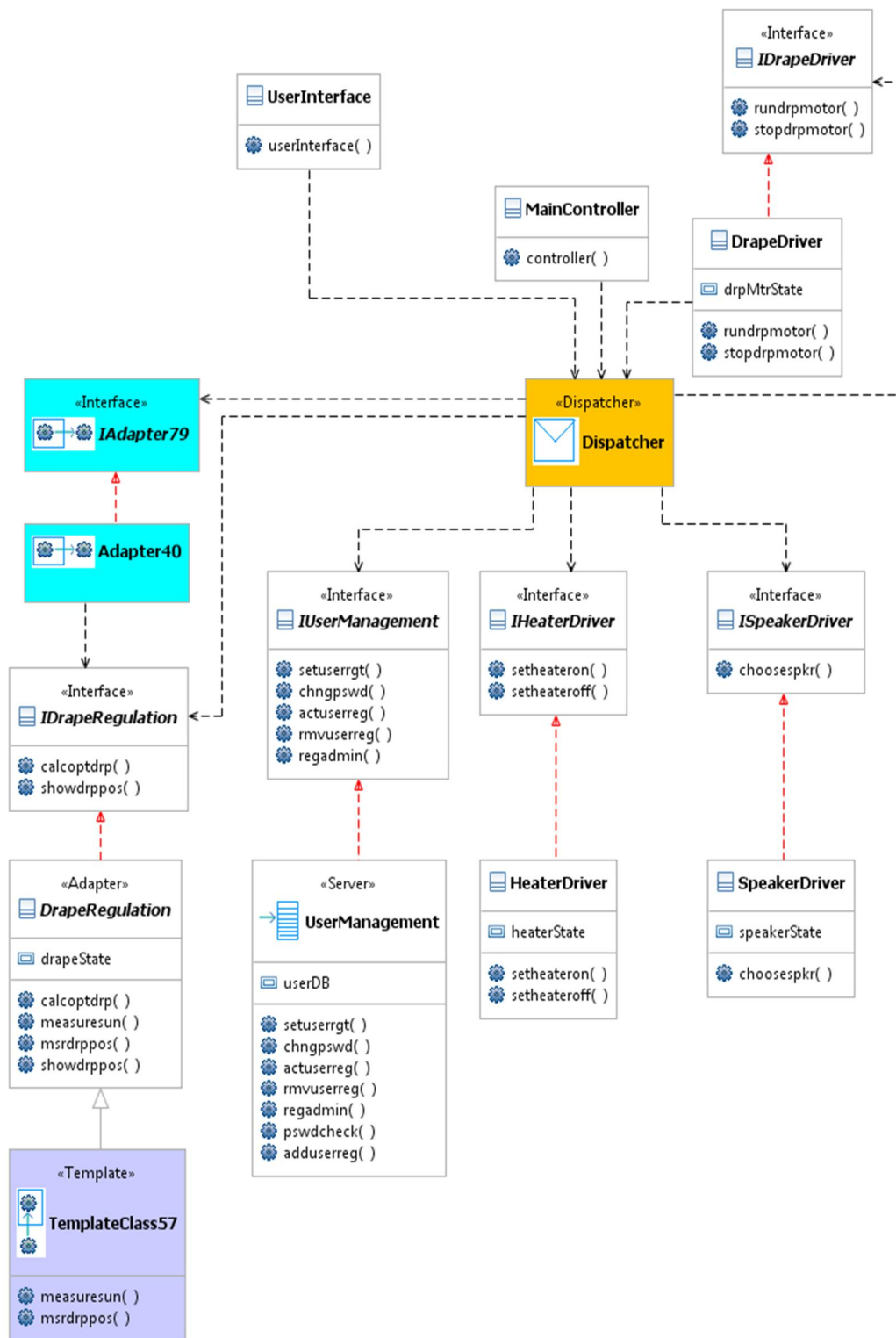


Figure 45. A fragment of proposed architecture for ehome

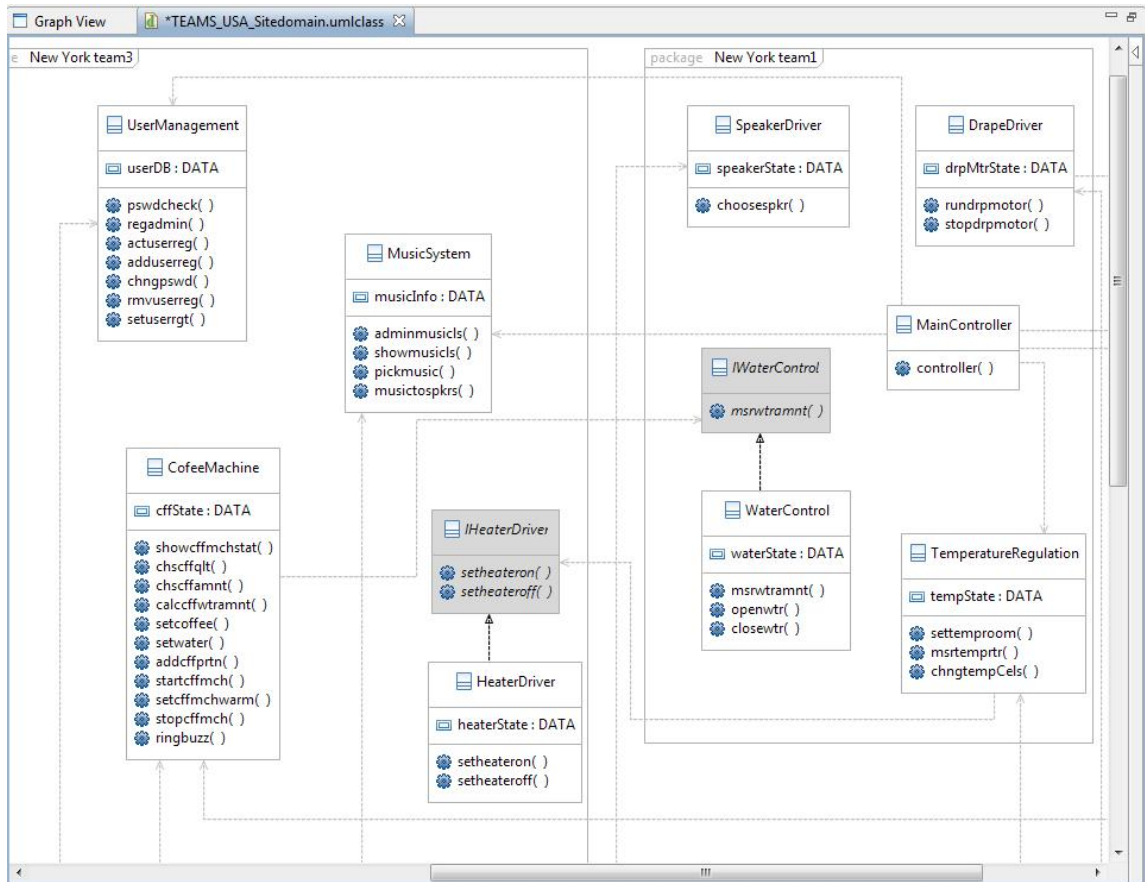


Figure 46. A fragment of proposed work distribution proposal for ehome

6.4 Summary

To summarize, Darwin tool eases the application of genetic algorithm-based approaches for software architecture design and project planning, as all the required information for applying the approaches can be expressed using the graphical user interface. The Darwin tool makes it possible to save, open and edit an evolution. It visualizes the representation of genetic algorithm execution through different views and contains different UML diagram editors for working with the result. Also, it provides the possibility to change the parameters of the genetic algorithm on the fly. This feature is especially useful for understanding how different input parameters influence the results. As UML diagrams are widely used in software industry, we anticipate that it would be easy for the practitioners to adopt the Darwin tool. Moreover, Darwin tool could be extended for solving other research problems as well, which are discussed in Section 10.2.

7 INTERLEAVING HUMAN DECISION MAKER AND AUTOMATED SUPPORT

The multiple studies conducted in this thesis have addressed the interleaving of manual and automated support [P3, P6]. This chapter presents two methods in which a human decision maker can collaborate with the automated support. The contents of this chapter correspond to the thesis contribution 5, i.e., “methods for collaboration of human decision maker and automated support”. This chapter is organized as follows. Section 7.1 introduces a semi-automated architecture design process, where a human decision maker can use her knowledge in guiding the Darwin tool towards better solutions. Section 7.2 presents a decision support mechanism, where the automated support provides a set of recommendations. Based on the recommendations, the project manager can decide on the optimal work distribution plan to be adopted. Finally, the chapter is summarized in Section 7.3.

7.1 Semi-Automated Architecture Design Process

Software architecture design involves many routine decisions, which an automated support can predict, but there are some sophisticated decisions that only a human architect can make. In practice, while designing architectures the architect makes a lot of decisions based on the tacit/implicit knowledge she gained over the years [Van Heesch and Avgeriou, 2011]. Thus, the benefits of automated and manual architecting can be combined by involving the architect in the automated architecture design process. This kind of interaction can enable the architect to use her knowledge in steering the genetic algorithm towards optimal architecture proposal. Moreover, this kind of interaction can improve the quality of the solutions, as it leverages humans ability in areas in which people currently outperform computers, such as strategic thinking, and the ability to learn from experience [Klau et al., 2010]. Such an interactive architecture design approach can be exploited in various development scenarios, such as incremental development of software architecture from scratch, the revision of an existing architecture due to changed requirements, etc. Here we will explore a scenario where software architecture is incrementally developed from scratch.

The overview of incremental architecture generation process is presented in Figure 47. The process starts with gathering requirements as use cases and creating the initial design. An initial architecture proposal is then generated by applying Darwin on the initial design (discussed in Section 6.3). Next, the architect can judge the quality of the architecture proposal and can make manual corrections to those parts of the architecture which are suboptimal. The modifications to the architecture can be related to introduction or removal of patterns from the architecture. In addition, the architect can freeze patterns that have to be taken into account by the genetic algorithm. As the genetic algorithm randomly inserts and removes patterns to and from the architecture

proposals, it is possible that certain patterns might not survive through the evolution process. To avoid this, the architect can freeze a pattern that has to be retained by the genetic algorithm. Similarly, the architect can also mark patterns that are not appropriate, which implies that the genetic algorithm will not apply those patterns in the automated architecture generation.

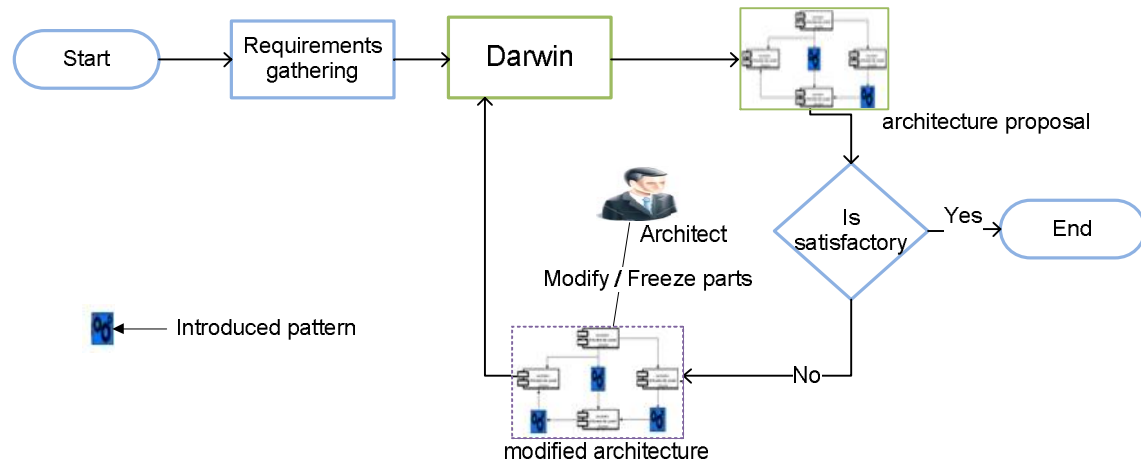


Figure 47. Incremental semi-automatic architecture generation

The modified architecture can be used as a seed to the Darwin tool for more fine grained improvements. The genetic algorithm improves the architecture without modifying the frozen patterns that are preferred by the architect. The resulting architecture proposals contain new patterns introduced by the genetic algorithm, together with frozen patterns. The manual modifications and automatic generation of design can be then repeated until satisfied architecture is produced. As can be seen, this process resembles the practical software architecture design process discussed in Subsection 2.4.

7.1.1 Tool Mechanisms for Interactive Architecture Design

To facilitate the interactive architecture design process described earlier, the class diagram view of Darwin (discussed in Section 6.1) has been extended to create an *architecture view*. In addition to creating classes and relationships (as in class diagram view), the architecture view can be used for introducing an existing architecture as input for Darwin. The existing architecture can be either a manually designed architecture proposal or an architecture proposal produced by the Darwin itself. Using the architecture view shown in Figure 48, the architect can construct the architecture incrementally, adding or removing some patterns after applying the genetic algorithm and continuing the automated architecture design process with the revised architecture, etc. This view alone provides the possibility to mix the architect's decisions into automatically generated designs.

Moreover, the architecture view contains user interface controls for freezing a pattern and for marking certain patterns in the architecture as unwanted. The *freezing patterns* mechanism allows the architect to fix the patterns. The genetic algorithm does not touch these patterns during the execution. For example, if a class introducing a Strategy pattern is frozen, then the operation associated with the Strategy pattern is not touched during the evolution process. However, in some cases, the genetic algorithm may propose solutions that are not appropriate in a certain context, i.e., on a subsystem or class. To avoid such inappropriate solutions *withdraw patterns* mechanism can be used. For example, if a

pattern introduced to a class is marked as withdrawn, then the genetic algorithm will not apply that pattern again for that class during the architecture synthesis.

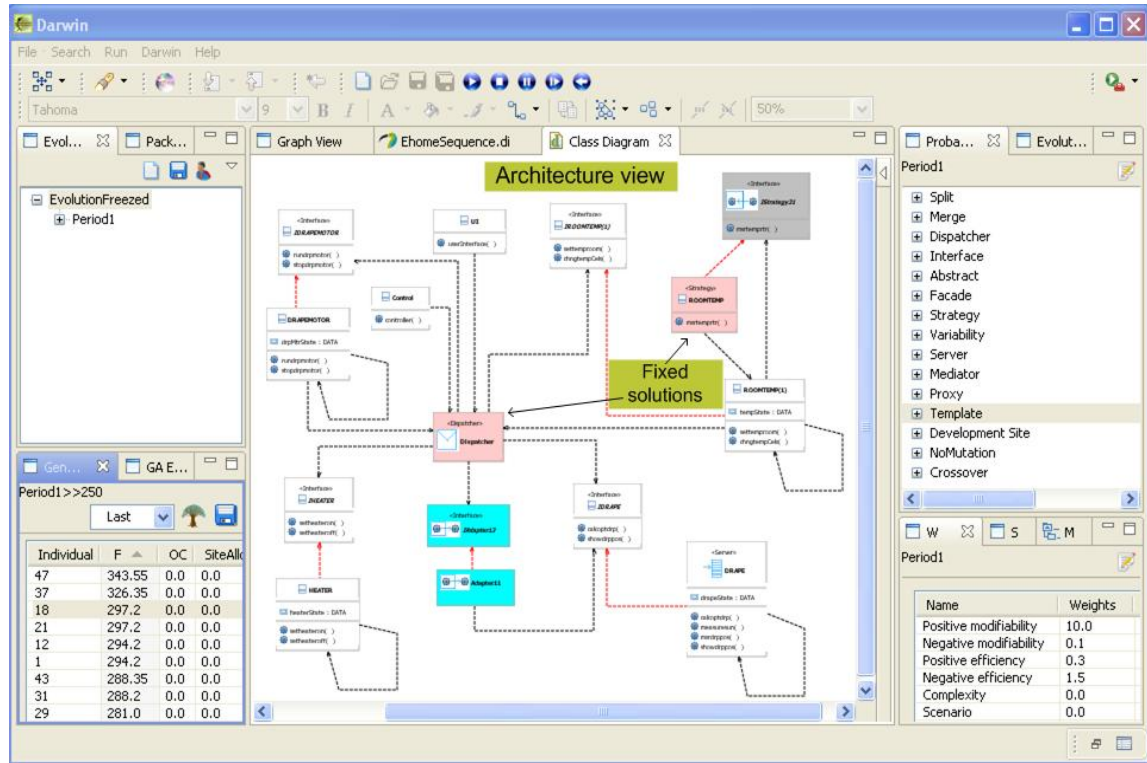


Figure 48. Darwin user interface with architecture view

7.1.2 Example

The semi-automated architecture design process is examined by designing the architecture of ehome system given in Appendix B. The initial design used in the example is shown in Figure 49 A (for presentation purpose only two subsystems of ehome are used). In the first iteration, the initial design shown in Figure 49 A is used as input for the Darwin. The architecture proposal produced after applying the genetic algorithm is presented in Figure 49 B.

As can be seen from Figure 49 B, the architecture proposal contains different design patterns and architecture styles. Next, the architect judges the quality of the architecture proposal and makes certain modifications using the architecture view of Darwin. The architect decides that it is sensible to use Message Dispatcher for other communication as well, as message based communication is already employed. The architect modifies the architecture such that class MainController also uses the Message Dispatcher for communicating with classes TemperatureRegulation and DrapeRegulation. After the modifications, the architect freezes the dispatcher connections between MainController and its client dependencies, so that they are retained in the resulting architecture proposal. In addition, to retain the Strategy pattern applied on the measureTemperature operation (msrtemptr in Figure 49 B) the architect freezes the Strategy pattern. The modified architecture proposal is shown in Figure 49 C. After the manual modifications, the genetic algorithm is again applied to the modified architecture proposal (shown in Figure 49 C) for more fine grained improvements. The architecture proposal produced by the genetic algorithm after second iteration is presented in Figure 49 D. As can be seen, the modifications (i.e., fixing Strategy and Message Dispatcher) specified by the

architect are retained by the genetic algorithm. In addition, new solutions (i.e., Adapter patterns) are introduced into the architecture. The process of modifying the architecture and applying the genetic algorithm for further fine tunings can be repeated until the quality of the architecture is sufficient.

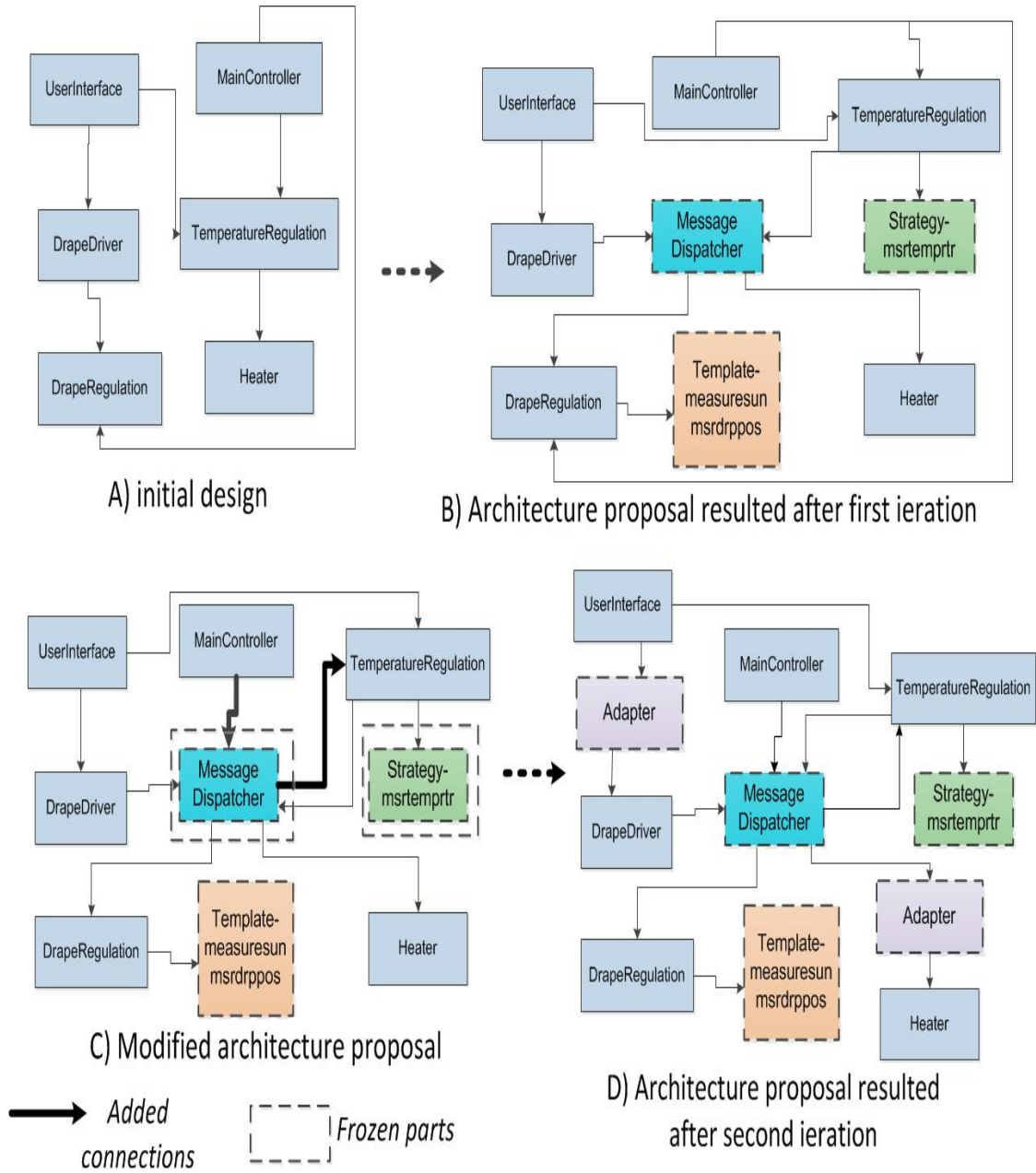


Figure 49. Interactive architecture design process with Darwin

7.1.3 Empirical Study

As shown in the previous subsection, the interactive architecture design process takes the architect decisions into account during the design process. However, an empirical study is needed to evaluate the quality of the resulted architecture proposals.

In the empirical study reported in publication [P3], an architecture proposal resulted from interactive architecture design process (described in the previous section) after three iterations is compared against two best architecture proposals designed by the

software engineering students. The example system used in the study is the e-home system discussed in previous subsection. The students were given the same set of information and solutions that are available for the genetic algorithm and are asked to design the architecture with an emphasis on the modifiability of the system. The average time spent by students for producing an architecture proposal is approximately 40 minutes. The time consumed for generating the resulted architecture proposal using semi-automated architecture design process was about 5 minutes, which includes the time consumed for modifying the architecture and freezing the decisions and executing the genetic algorithm. The architect has made approximately two to three modifications and has frozen four decisions during the interactive architecture design process. The decisions were made in order to produce modifiable architectures. Moreover, during the interactive architecture design process, the architect can only modify the architecture using the solutions that are available for the genetic algorithm and students.

The architecture proposals from students (named S1 and S2) together with the architecture proposal (named R) resulted from interactive architecture design process were presented for three software engineering experts for evaluation. The experts were unaware of synthesized architecture and manually designed architectures. The experts were then asked to order the solutions according to the overall quality of the architecture. The results from the experts are shown in Table 7. As can be seen, all the three experts have ranked the solution R generated from interactive architecture design process as better solution than the solutions S1 and S2 designed by software engineering students.

The approach is still in its infant stage, but the empirical study shows that with very little human interaction, the genetic architecture synthesis approach can produce architecture proposals that can be comparable to architecture proposals designed by senior software engineering students. The approach is efficient, as it has taken a fraction of time spent by the students for designing the architecture. One other advantage of the approach is that it can suggest patterns in those places, which a human architect has never thought of. Moreover, the approach provides a way to utilize architect knowledge to drive the search algorithm towards good solutions, as well as to make the architects trust the solution produced by the automated approach.

Table 7. Results of experts evaluation

<i>Experts</i>	<i>Order of solutions according to their quality</i>
Expert1	R, S1, S2
Expert2	R, S2, S1
Expert3	R, S1, S2

7.2 Interactive Support for Work Distribution Decisions

Project planning is carried out in early stages of software engineering life cycle, where most of the information is based on estimates that are unreliable. In these situations, instead of presenting actual solutions, the automated support should provide a set of solutions giving insight to how different management selections and estimates affect the

solutions. These solutions can be then used by the project manager in deciding on the final solution.

To facilitate interactive decision support, the fitness graph view of Darwin (discussed in Section 6.1) has been extended to create a *multi-objective fitness graph view*. The multi-objective fitness graph view shows a set of Pareto optimal solutions across the cost and duration space, as shown in Figure 50. As can be seen, each solution shown on the graph is no worse than any of the other solutions, but also cannot be said to be better with respect to both objectives.

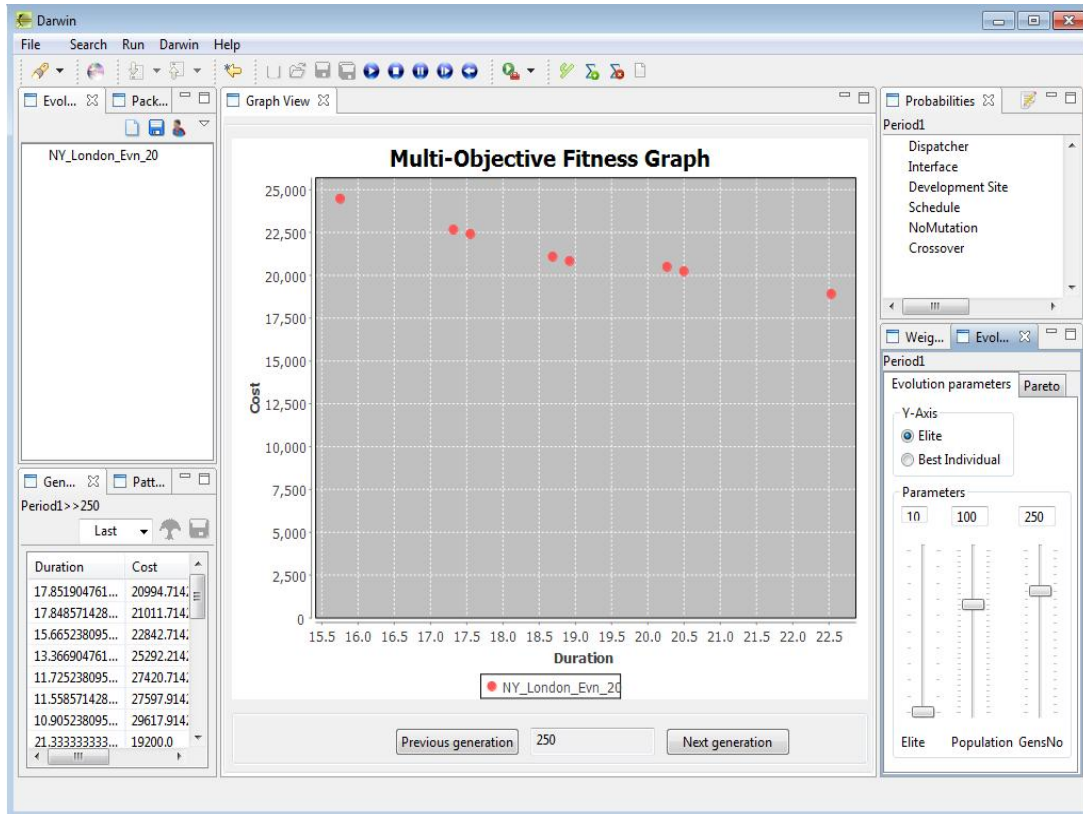


Figure 50. Multi-objective fitness graph view of Darwin

Using the multi-objective fitness graph view, the project manager can select suitable work distribution for the project at hand. Figure 50 shows an example of a multi-objective fitness graph obtained for distributing work to New York and London teams, as discussed in Section 4.3. For example, if the goal of the project is shorter completion time, then the manager can choose a solution (named P1) achieving shorter completion time (around 16 days) with high cost (around 25000\$). Alternatively, if the goal is low development cost, then the decision maker can choose solution (named P2) achieving low development cost (around 18000\$) with high completion time (around 22 days).

Moreover, the project manager can examine the work distribution proposals of different solutions on the graph (by clicking the dots) for understanding how different work divisions have influenced the cost and duration estimates. For example, the manager can check the work distributions of solutions P1 and P2 to analyze the reason for high cost in solution P1 compared to solution P2. The work distribution proposals of solutions P1 and P2 are shown in Figures 51 and 52.

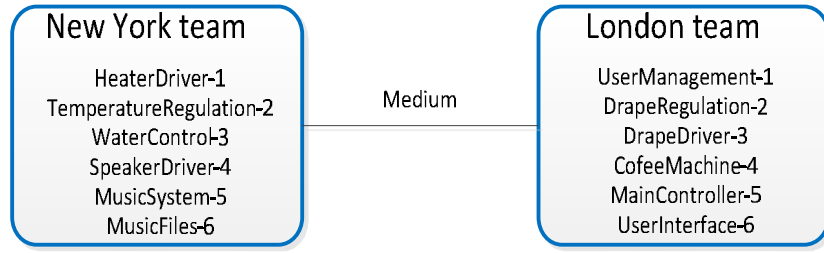


Figure 51. Work distribution proposal of solution P1

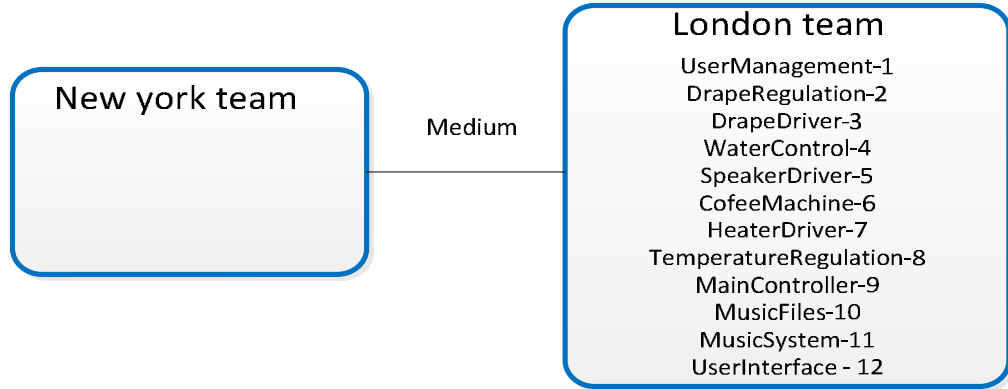


Figure 52. Work distribution proposal of solution P2

In the work distribution proposal of solution P1, both teams New York and London are used for developing the work. This shows that the use of both the teams have resulted in higher cost in solution P1 compared to solution P2, where all the work is developed at one site, i.e., London team. However, the completion time is higher in solution P2 compared to solution P1, as the amount of effort spent on the project per day is less in solution P2 compared to solution P1. In this way, the project manager can examine different solutions on the graph and can decide on the suitable work distribution according to the project goals.

7.3 Summary

This chapter presented two methods for the human decision maker to collaborate with the Darwin tool. In the first method, the software architect can introduce his decisions into the automatically produced architecture proposals and can again apply the genetic architecture synthesis approach. The tool considers the decisions made by the architect and tries to improve the suboptimal parts of the architecture proposal. In the empirical study conducted with the approach, we noticed that the produced results are on the similar standard as the design proposals created by the senior software engineering students.

In the second method, the project manager can interactively go through the solutions in the Pareto front and can analyze how different solutions influence the cost and duration of the project. This kind of analysis would be helpful for the decision maker, as it would give more insight into how prioritizing or constraining different factors will influence the project goals. Moreover, we noticed that these collaborative methods could be applied to new research problems as well, which are briefly discussed in Section 10.2.

PART IV – CLOSURE

This part discusses the related work and limitations of the study. The research contributions are also summarized in this part. The future research directions are described and finally the part is concluded with final remarks.

8 RELATED WORK

The objective of this chapter is to present existing work related to the main contributions in this thesis. This chapter is organized as follows. Section 8.1 presents studies that apply genetic algorithms and constraint satisfaction techniques for automatically designing the software architecture of a system. Section 8.2 discusses studies using custom initial population in genetic algorithms for solving different problems. Finally, in Section 8.3 the studies that support work distribution in GSD projects are discussed.

8.1 Software Architecture Design

The amount of studies that aim to automatically discover an optimal software architecture design with respect to a set of quality requirements have proliferated in the last decades [Aleti et al., 2013; R  ih  , 2010]. Similar to our approach (discussed in Section 3.1), the majority of the approaches use meta-heuristic search algorithms as an optimization technique. However, the common denominator for these approaches is that they are targeted at improving an existing architecture rather than create it from *requirement-level information* (i.e., initial design and quality attributes weights). This section first presents the studies that apply meta-heuristics for software architecture design, followed by studies using constraint satisfaction techniques for software architecture design.

8.1.1 Studies using Meta-heuristics for Software Architecture Design

Amoui et al. [2006] have studied the application of genetic algorithms for increasing the reusability of software by applying architecture design patterns to a UML model. The aim is to find sequence of pattern transformations that improve system reusability. In the genetic algorithm, they used design patterns specified by Gamma et al. [1994]. Their approach focuses on only one objective, i.e., reusability of the system, whereas the genetic algorithm used in this work aims at optimizing multiple qualities of software architectures. Moreover, the input used in their approach is an existing architecture that is more elaborated than the requirement-level information used in the genetic architecture synthesis approach.

O’Keeffe and O’Cinneide [2004] applied simulated annealing for improving a design with respect to a conflicting set of goals and developed a tool named *Dearthoir*. The algorithm restructures a class hierarchy and moves operations in it in order to reduce methods that are inherited from the super classes but are not used in the inherited classes. In addition, the algorithm also aims to eliminate code duplications and ensures that super classes are abstract when appropriate. They have continued their research by studying refactoring in object oriented programs [O’Keeffe and O’Cinneide, 2006; O’Keeffe and O’Cinneide, 2008]. They have developed a *CODE-Imp* tool for refactoring object-oriented programs such that they conform more closely to a given design quality model. However, in this thesis work, the operations are already grouped into classes. Moreover, instead of refactoring the existing architecture, the genetic algorithm

presented in this work aims at improving the overall quality of the architecture by applying well-known patterns. Also, the quality metrics used in their work are simpler than the quality metrics used in this thesis work.

However, instead of refactoring the architecture, the genetic algorithm presented in this work aims at improving the overall quality of the architecture by applying well-known patterns. Also, the quality metrics used in their work are simpler than the quality metrics used in this thesis work.

Seng et al. [2005, 2006] have applied genetic algorithms for improving subsystem decomposition of a software system. For a given subsystem decomposition, the genetic algorithm tries to find a subsystem decomposition that optimizes the fitness function. The fitness function is based on the coupling, cohesion and complexity metrics as well as heuristics, such as cyclic dependencies and bottle necks. Similarly, Mancoridis et al. [1998] have presented a *Bunch* tool for automatically partitioning the components of a system into subsystems based on coupling between them. The bunch tool creates a hierarchical view of the system based on the components and relationships that exist in the source code. The Bunch tool uses hill climbing and genetic algorithms to find proposals that minimize inter-connectivity and maximizes intra-connectivity between subsystems. Mitchell et al. [2002] have built *ARIS* tool on top of the Bunch tool. Software developers can use ARIS to specify rules that govern how modules and subsystems can relate to each other. ARIS then provides assistance in identifying the violation of such rules. There are multiple differences between these approaches and our approach. First, they operate on subsystems, modules and source code, whereas our genetic algorithm operates on classes, operations and solutions. Second, their main focus is on reverse engineering an existing mature system, rather than designing the architecture from requirement-level information, as carried out in this thesis.

Di Penta et al. [2005] developed a tool kit, software renovation framework (SRF), to cover several aspects of software renovation, such as refactoring existing libraries into smaller ones and for removing unused objects and clones. The SRF framework first identifies the dependencies between the artifacts and then applies genetic algorithms and hill climbing for refactoring. Also, the feedback from developers is taken into account in refactoring. The SRF framework focuses on reverse engineering and the metrics used in the fitness function are simpler than the metrics used in this work.

Martens et al. [2010] have presented an approach to automatically improve a given architecture model with respect to performance, cost and reliability. The approach is implemented in a tool named PerOpteryx. As input, the tool requires a component-based architecture model with cost, performance and reliability annotations. The tool uses genetic algorithm for finding Pareto optimal candidate solutions. The focus of this work is different from ours. Their work attempts to improve an existing architecture rather than designing the architecture from requirements, as carried out in this thesis. Also, the objectives (e.g. processor speed and number of servers) optimized in their work are quite different from ours.

Aleti et al. [2009] have attempted to optimize the architecture with respect to data transfer, reliability and communication overhead. They have presented ArcheOpterix tool that provides support to implement different architecture evaluation and optimization algorithms. ArcheOpterix uses a genetic algorithm with Pareto optimality for finding best solutions. Compared to our approach, their focus is not on the actual

architecture design but on the optimal deployment of software components to a given hardware platform. Also, the metrics used by them are quite different compared to our metrics.

Simons et al. [2010] have applied evolutionary, multi-objective search and software agents to aid the architects in class design. The starting point for the approach is use cases. Data (nouns) in the use cases are transformed into attributes and actions (verbs) are transformed into methods. The attributes and methods are then grouped into classes. Each class should have at least one method and one attribute and no method or attribute can be in more than one class. One individual is the design containing all methods and attributes and their class distribution. Mutation results in moving a set of methods or attributes to different class and crossover results in swapping of attributes and methods of two classes. The fitness of individuals is calculated using coupling and cohesion between the classes. During the evolutionary search, a human designer interacts with the search algorithm to steer it towards interesting class designs. The authors have further extended their work by investigating the role of elegance and symmetry in the produced software designs [Simons and Parmee, 2012]. In searching for the elegant designs, the approach considers both the human designer judgement and fitness computations made by the search algorithm into account. Similar to our work, this approach also starts from gathering use cases, but their main focus is to optimize the assignment of methods and attributes to classes and interaction between classes. In this thesis work, however, the operations and attributes are already grouped into classes in the initial design given as input to the approach. Also, the proposed genetic algorithm focuses on improving the overall quality of the system by introducing different patterns rather than moving operations between the classes. Moreover, the human designer does not have ability to alter the designs during the evolutionary process, as described in Section 7.1.

In addition to evolutionary algorithms, the authors have applied other search algorithms like ant colony algorithms to the class design problem [Simons and Smith, 2013]. With the ability of ant colony algorithms to provide good results in reasonable time, the authors have proposed interactive ant colony optimization approach to the class design problem [Simons et al., 2014]. Moreover, for gaining insights into how scale and complexity influences the software design search space, the authors have further extended their work by applying a self-adaptive mutation [Smith and Simons, 2015]. Also, the authors have examined how different factors involved in the search algorithms influence the class design problem. The authors have continued their research by investigating how to present solutions to the user for accurate evaluation in the interactive meta-heuristic search and proposed preference meta-heuristic design pattern [Aljawawdeh et al., 2015].

Bowman et al. [2010] have applied multi-objective genetic algorithm for solving the class responsibility assignment (CRA) problem. It is similar to the class design problem studied by Simons et al., as it attempts to assign responsibilities to classes. The aim is to provide interactive feedback to the designer than at producing a whole design. The fitness of the solution is measured using coupling and cohesion metrics. As an input to the algorithm, a class diagram together with constraints on what can and cannot change in the class diagram is given. The genetic algorithm evaluates the class diagram and proposes possible improvements. Similarly, Glavaš and Fertilj (2011) applied meta-heuristics for solving the CRA problem. Their main objective was to compare different meta-heuristic algorithms to specify which one is more suitable to solve CRA problem. They found that simulated annealing, hill climbing, particle swarm optimization and

genetic algorithm produced better results than the random search. The focus of these approaches is different from ours. They attempt to optimize an existing architecture rather than designing the architecture from requirement-level information, as carried out in this thesis.

Li et al. [2011] have proposed *AQOSA* (Automated Quality-driven Optimization of Software Architecture) toolkit for automatically improving the quality properties of component-based software architectures. The toolkit makes use of the evolutionary multi-objective algorithms, such as Non-dominated Sorting Genetic Algorithm II (NSGAI) and Strength Pareto Evolutionary Algorithm 2 (SPEA2) for generating alternative architecture proposals. They have continued their research by integrating problem-specific knowledge i.e., architecture design patterns and antipatterns into the architecture optimization problem [Etemaadi et al., 2012]. There are multiple differences between this work and our work. Firstly, AQOSA is applied for improving an existing architecture, rather than designing the architecture from requirement-level information, as in our case. Secondly, the quality attributes used in their work are different from ours. Their focus is on improving quality attributes like processor utilization, architecture cost and security.

Similar to the Darwin tool presented in Chapter 6, *ArchE* (Architecture Environment) [Mcgregor et al., 2007] provides support for interactively designing the architecture of a system. As input, ArchE takes quality attribute requirements and the set of features that the system should support. In addition, if a legacy design is available that can also be given as input to the tool. The tool then identifies the dependencies between the requirements. The architect can interact with ArchE to further identify the dependencies among the requirements and to provide properties that are required in order to predict quality attribute behavior. ArchE then creates an initial architecture incrementally with a series of suggestions. If the architect accepts a suggestion, then ArchE applies it to the architecture, calculates the effects of the revision, and shows the revised information. The interaction/revision continues until the architect is satisfied with the design. However, when compared to Darwin, the main difference with ArchE is that it uses deterministic search for performing architectural modifications. Compared to ArchE, the Darwin tool should provide much more inventive solutions, as it is not required to follow a pre-defined path in choosing the solutions, and can search for solutions from much wider spectrum.

8.1.2 Studies using Constraint Satisfaction Techniques for Software Design

There are multiple studies (e.g. [El-Boussaidi and Mili, 2008], [Gueheneuc et al. [2001]]) on applying constraint satisfaction techniques for solving problems related to software architecture design. However, most of them are related to identifying design patterns from an existing model, rather than creating software architecture from the requirement-level information of the system.

The work of El-Boussaidi and Mili [2008] has used constraint satisfaction techniques for identifying instances of the design patterns in the input model. The problem of identifying design pattern instances is modeled as a CSP. After identifying the design patterns the input model is transformed according to the pattern identified. Similarly, Gueheneuc et al. [2001] have used CSP's to correct inter-class design defects in the source code. They have defined a meta-model for representing design patterns. Following the meta-model the abstract models of design patterns are defined. An

abstract model of a design pattern describes the entities of a design pattern and their relationships. Moreover, the source code is also modeled according to the meta-model. The source code is then analyzed to identify the patterns matching the abstract models. If the abstract model of a pattern is matched to some structure with some relationships missing then the differences and options to correct them are suggested. In addition, they have also developed a prototype tool *PTIDEJ* for suggesting the corrections. There are several differences between the approach presented in Section 5.2 and these approaches. Firstly, their focus is on reverse engineering, which is different from our focus, i.e., designing a system from functional requirements. Secondly, their input model is more elaborated than our initial design.

Mederly et al. [2010] have used CSP's for automatic design of messaging-based integrated solutions. The approach takes the abstract representation of the integration problem as input, which also contains information on business services that are incorporated into the solution, logical flow of messages among these services and a set of requirements (e.g. throughput and reliability for each business service, supplicate message avoidance) to be met. The method solves the integration problem by choosing appropriate communication channels (e.g. publisher/subscriber, point to point [Hohpe et al., 2004]), inserting appropriate integration services (e.g. Message Translator, Message Filter [Hohpe et al., 2004]) and deploying components appropriately. Castro et al. [2008] have applied CSP's to automate the generation of composite COTS-based software systems for the given functional and non-functional requirements (e.g. cost, reliability). The main idea for using CSP's is to explore different solution alternatives and choose a solution which best meets the requirements. The similarity between these approaches and our approach is that constraint satisfaction techniques have been used for automatically solving a complex software engineering problem. However, the focus of these approaches is on application integration and constructing a system from existing components, which is different from our focus, i.e. designing architecture from requirement-level information.

8.2 Population Initialization in Genetic Algorithm

Several studies like [Julstrom, 1994], [Cochran et al., 2003] and [Morrison, 2003] have used the idea of seeding the initial population of genetic algorithm for solving different problems. However, to the best of our knowledge, there exists no study that uses a similar population initialization method as described in the quality farms approach (discussed in Section 5.1) for solving a multi-objective problem.

Cochran et al. [2003] have described a multi-population genetic algorithm (MPGA) to solve multi-objective scheduling problems for parallel machines. The MPGA algorithm operates in two stages. In the first stage, a combined fitness function is used for evaluating the solutions. In each generation of the first stage, the top solution for each single objective, as well as the top solution for the combined objectives, i.e., with respect to fitness function is stored. The collected individuals from stage one are rearranged into separate (sub) populations and are used as initial population for stage two. Each sub-population is then evolved individually while an elitist strategy preserves the best individuals of each objective and the best individual of the combined objective. When compared to their method, our quality farms approach does not separate population after the first stage. Moreover, compared to our approach, very limited number of individuals is collected after first iteration. Also, no crossbreeding operator is applied on the individuals collected after the first stage.

Similar to the quality farms approach, Rahnamayan et al. [2007] have studied a novel population initialization method for accelerating the genetic algorithms. The approach first generates a random initial population. Next, for every chromosome in the initial population an anti-chromosome is generated by inverting all the bits of the chromosome. Finally, the fittest individuals from both the initial population and anti-chromosomes are selected as initial population. Compared to our approach, the approach does not involve any crossbreeding between the individuals and there was also no consideration of the multiple objectives in creating the initial population.

Schaffer [1985] has described a vector-evaluated genetic algorithm (VEGA) for multi-objective optimization. The approach modifies the selection mechanism in such a way that after each generation a number of sub-populations are generated for each problem objective. For a problem with K -objectives, K sub-populations of size N/K are generated (where N is the size of total population). These sub-populations are then shuffled together to obtain a new population on which the crossover and mutation operators are applied. This is performed to achieve crossbreeding between individuals from different population groups. However, the actual crossbreeding between the individuals rarely happened, as the genetic algorithm tended to move towards individuals that are good in one objective and not at all good in other objectives [Murata et al., 1996]. Moreover, the quality farms are more refined than their sub-populations, as the farms are developed until the end of an evolution period compared with their sub-populations, which are obtained after each generation. Also, in our quality farms approach there is less risk to end up in similar extremes as in VEGA, since after crossbreeding all the individuals are evaluated similarly.

8.3 Project Planning

Several new studies applying meta-heuristic search algorithms for planning software projects have been published in the last decade [Ferrucci et al., 2014; Peixoto et al., 2014]. The most studied problems are assigning tasks to employees while taking into account their skills and attempting to simultaneously optimize the cost and duration of the project [Chang et al., 2001; Alba et al., 2007; Minku et al., 2012; Debels et al., 2005]. On the other hand, there are also studies on optimizing the estimated effort required to complete a project [Dolado and Fernandez, 1998; Shukla, 2000], and on defining the cost function of software projects [Dolado, 2001; Ferrucci et al., 2010; Sarro et al., 2012]. However, the majority of approaches does not consider the projects in a globally distributed setup, which differentiates them from our approach presented in Chapter 4. This section briefly presents the approaches that address problems related to project planning in GSD and have similar goals as ours.

The study by Fernandez et al. [2012] has applied genetic algorithms for task allocation in GSD projects. The approach takes list of tasks, available resources, goals and different parameters used to characterize these entities as input. Based on the cost and quality goals, the suitability of each resource to perform the task is computed. However, compared to our planning approach (discussed in Chapter 4), this approach did not consider organizational aspects, such as communication structure and characteristics of the teams. Moreover, as the approach was not described in detail, it remains unclear how the approach was modeled, evaluated, and what are the advantages and disadvantages of the approach.

Mockus et al. [2001] presented a simple model for task allocation in global software development process. The model is based on the assumption that software development can be described as a series of modification requests to a set of modules. A modification request is a set of changes to the existing code files and a module means a set of code contained in the directory of files. Based on the model, an algorithm is developed for finding an optimal assignment of sites to modules. The algorithm takes a set of modules and modification requests as input and then iteratively transfers modules to new sites with the goal of minimizing the modification requests spanning multiple sites. The algorithm uses a variation of simulated annealing. The main difference between this approach and our planning approach is that they consider only one single objective, i.e., minimization of communication, but they do not take into account other influential factors and project goals like cost and duration, which limits the adaptability of the approach to other project goals.

The work of Setamanit et al. [2007] applied discrete-event and system-dynamic simulation for developing a simulation model, which they used for evaluating different task allocation strategies. The model simulates software development at each site, effect of interaction between sites and can make statements about effect of different strategies on the productivity. However, compared to our model, they do not consider dependencies between the tasks. Also, their goals such as defect and effort rate are different from our goals cost and duration.

Lamersdorf [Lamersdorf and Munch, 2010; Lamersdorf, 2011] have developed a multi-criterion distribution model for task allocation in GSD projects. The criteria and causal relationship of the model were identified by conducting a literature survey and was refined in a qualitative interview study. The approach takes information about the tasks to be distributed, site information and different parameters of these entities and suggests work allocation based on cost, quality and time. They also reported a tool *TAMRI* [Lamersdorf and Munch, 2009] for distributing tasks in GSD projects. The tool makes use of the multi-criteria model for assigning tasks to distributed teams. Their work is closest to our work with respect to the goal. The major difference between our approach and their approach is the algorithm used for identifying the work assignment. They have used distributed systems approach with Bayesian networks for identifying the work assignments. In our case, the genetic algorithm was used for performing the work assignment. Compared to their approach, our approach can propose a set of Pareto optimal solutions on the cost and time space. The project manager can then explore these solutions and can choose the solution suitable for the project at hand.

Almeida et al. [2011] have studied multi-criteria decision analysis for planning GSD projects with Scrum. They reported a model for project managers to make decisions using cognitive mapping [Kitchin, 1994] relying on user input. The application of the model can propose suggestions, such as where the product owner has to be, where scrum master could stay, etc. However, their approach does not talk about how to assign work to teams. Moreover, when compared to their approach, we take a step further to the field of automated project management, considering also business aspects.

Mak and Kruchten [2007] reported *NextMove* framework for task coordination in distributed agile software development. The framework assists project managers in answering questions like what should be done next and who should do it. The tool uses multi-criteria decision resolution methodologies for making the decisions based on the team member attributes, project schedule and feature priorities. The objective of the tool

is to find a suitable developer based on the work load and past experience. However, the approach does not consider the communication problems caused due to the work distribution. Moreover, their focus is on feature release, which is different from our focus, i.e., task allocation in GSD projects.

Yildiz et al. [2012] have proposed a tool support for aiding project managers in planning GSD projects. They presented a framework including meta-model for deriving application architecture for GSD projects. The tool presents the project manager a set of questions related to the employees available for the project, their skills, site information, work cultures, etc. Based on the information from the project manager, the tool proposes a model to carry out the project. The model provides guidelines, such as how to deploy teams in different sites, how to carry communication between the teams, tools for communication, etc. There are several differences between their work and our work. Firstly, the model does not make decisions related to work assignment to the teams. Second, their approach does not consider the cost and duration aspects of the project, which are essential for planning. Moreover, their focus is on deriving the application architecture to carry out a GSD project, which is different from our focus, i.e., work distribution in GSD projects.

As described in Section 4.3, software project planning is a multi-objective problem, where multiple competing objectives have to be specified. In order to match the multi-objective nature of project planning, several researchers have applied Pareto optimality to study project management problems [Sayyad et al., 2013]. The majority of them are concerned at solving the task allocation and scheduling problem [Duggan et al., 2004; Chicano et al., 2011; Antoniol et al., 2011], agile team allocation problem [Britto et al., 2012], team staffing problem [Stylianou and Andreou, 2013] and overtime planning [Ferrucci et al., 2013]. However, none of these approaches consider the projects in globally distributed setting, which differentiates them from our work.

9 LIMITATIONS

The main aim of this research is to assess the applicability of genetic algorithms to develop automated support for software architecture design and project planning activities. This chapter summarizes the main limitations of this research.

9.1 General Limitations

The main limitations of the experiments conducted in this study are presented in this section, including the threats to validity. According to Wohlin et al. [1999] the threats to validity that may affect an experimental study fall into four categories: conclusion, construct, internal and external threats.

The main limitation of this study is the difficulty to generalize the results because of the limited amount of the used example systems. In this study, a limited set of example systems were used for conducting the experiments. However, it should be noted that it is quite difficult to gather the realistic information from the industrial projects. The scale of the example systems used in the experiments is another concern. Although, the example systems employed in this thesis have different characteristics, it is difficult to generalize the results to larger systems. Obviously, conducting experiments using large example systems, preferably from industry, would have increased the level of confidence in the evaluation and further confirm the results. Indeed, the inclusion of more example systems would have increased the value of the proposed approaches. It is not a simple task to find the real and interesting projects from the industry. For many companies, the projects tend to be strategic and in many cases they do not allow a research to collect the required data.

Conclusion threats are concerned with whether the conclusions reached in the study are correct. The conclusion threats in search-based techniques experimental studies include not considering the random variation in the search and poor summarization of the data (Barros and Dias-Neto, 2011). In this thesis work, these threats are addressed by executing the genetic algorithm for multiple times (10 – 50 times) in all the experiments [Arcuri et al., 2011; Harman et al., 2012]. The multiple runs ensure that the generated results are statistically significant and are not due to the effects of random variation.

In the case of the reported experimental study discussed in Subsection 7.1.3, only two manually designed solutions were selected for evaluation. However, no control was applied in the experiment and all solutions presented to the experts were edited in such a way that it is difficult to distinguish the synthesized architectures and manually designed architectures. Taking a larger number of solutions would have resulted in a more fine grained comparison between the manually designed solutions and semi-automatically produced solutions. The chosen solutions are best solutions among several architecture proposals developed by senior software engineering students and it would be unlikely that choosing more student solutions would alter the outcome of the result. Also the

students were limited to a set of design choices for designing the architecture. Increasing the amount of design choices available for the students would further influence the results. This limitation is due to the patterns used in the Darwin tool and could not be avoided. One other limitation is that student architecture proposals were compared to the architectures modified by the experts in the interactive architecture design process. In future study, this limitation can be avoided by using students in the interactive architecture design process. Moreover, using more experts in the study would have strengthened the results. Despite that, it is fair to say that the results of the experts were consistent, as all the three experts have considered the synthesized solutions better than the manually designed solutions.

Internal threats evaluate if the relationship between conducted experiments and outcome is causal or results from factors which the researcher cannot control. In case of experimental studies in search-based techniques, major internal threats include poor parameterization, lack of a detailed description about the applied parameters. In this study, we have described the parameters used in the experiments and also details about how the example systems are built in Appendix A and B. It should be noted that the applied parameters, such as mutation and crossover probabilities, population size and generation size influence the efficiency of the genetic algorithm. Mutation and crossover probabilities influence the rate at which population evolves. If the probabilities are not selected properly, there is a risk that either populations will evolve too slowly or the genetic algorithm will end up searching aimlessly in the search space [Mitchell, 1996]. Similarly, the generation size and population size used in the genetic algorithm can greatly affect the outcome. To find the correct parameters for a problem category, we started genetic algorithm with some initial parameters and performed some trial and error iterations until results are satisfactory, which is the common way to choose parameters when applying genetic algorithms for a new problem. Moreover, these values may not be suitable for the new problem categories and need to be fine-tuned.

Construct threats are concerned with the relation between theory and observation. They ensure that the treatment reflects the construct of the cause and that the outcome reflects the construct of the effect. In case of search based techniques experiments, major construct threats involve not discussing the underlying model subjected to optimization. On regards to these issues, the modeling of the approaches and the applied fitness functions are discussed in detail in Chapters 3, 4 and 5.

External threats are concerned with the generalization of the observed results to a larger population, outside the sample instances used in the experiment. Major external threats to search based techniques experiments include the lack of a clear definition of target instances and not having enough diversity in instance size and complexity. In this study two example systems of different sizes and complexity are used. These instances were described in detail in the study. Moreover, an initial version of the Darwin tool and applied example systems are made available for further evaluation at [Darwin tool, 2016].

9.2 Software Architecture Synthesis

In the current stage, the software architecture synthesis described in this thesis has many limitations. The approach is limited by the availability of patterns. The patterns applied for the architecture synthesis are fairly limited and may not be suitable for producing completely realistic architectures. However, we expect that the growing increase in the

application of pattern mining process to the new domain areas could reduce this limitation [Eloranta et al., 2014]. The architectures are evaluated from limited quality attributes and many other quality attributes (e.g. reliability, security and usability) are not considered. This limits the applicability of the approach to real world systems. This limitation could be reduced by increasing the pattern base available for the design and by taking different quality attributes into account while evaluating the architecture.

The used input, i.e., initial design of the system may influence the results, as the proposed approaches were not able to alter the initial design in any other way than by applying patterns. For a given initial design, the final proposal will be good according to the specified fitness function. Moreover, the estimated support data of operations, such as frequency of use, variability and parameter size may influence the results, as the estimations may deviate from architect to architect, based on the architect's experience in the domain of the system. The estimated values are used by the fitness function in order to have more accurate fitness values. The effect of these estimations could be visible at some level in the architecture.

The applied fitness function may also influence the produced results. The fitness function evaluates the goodness of the architecture proposal and is the key for producing quality architecture proposals. The formulation of the fitness function may be biased towards the expertise of the fitness function designer related to the software architectures. However, the use of widely used quality metrics has decreased this limitation. This limitation is further reduced with the application of widely used solutions for designing the architecture, as the influence of these solutions on the quality of the system is well known.

In the case of the experiment related to constraint-based architecture design approach, the applied patterns are limited and the used objective function was very simple and may not be suitable for producing realistic architectures. This limitation could be reduced by extending the patterns available for the design and by considering different quality metrics in constructing the objective function. Moreover, it is difficult to know all the constraints related to the system in advance. This difficulty can be overcome by introducing a human architect into the constraint-based architecture design, as presented in the case of interactive search-based design process discussed in Section 7.1. For example, the architect can iteratively specify new constraints, freeze some solutions associated to the variables, and can call the solver to improve the design.

The quality farm approach has been applied to only two objectives. In the case of more than two objectives, the crossbreeding scheme has to be modified in such a way that the influence of all the objectives is taken into account in creating the initial population for the farm stage. However, further work is needed to evaluate how the genetic algorithm performs in optimizing more than two objectives using the farm-based approach.

9.3 Work Distribution in GSD Projects

The produced work distribution and schedule plans are of preliminary nature and the inclusion of more information about the teams would produce more practical work distribution and schedule plans. Naturally, the inclusion of more attributes would produce more practical work distributions. However, the experiments show the potential of the genetic algorithms for distributing work to GSD projects and for studying different planning situations in the context of GSD.

The used team structures may influence the results, as different project managers can estimate the characteristics of the teams and communication distance between the teams differently. This limitation can be reduced by considering the data from past projects to estimate the characteristics of teams and their communication distance. However, unless the estimated values are modified so that they can clearly show the differences between the teams, the proposals would not be dramatically different. Moreover, the fitness function used for calculating the duration and cost is very abstract and may not be sufficient for work distribution in practice. This limitation can be reduced by employing data behind widely used algorithmic cost estimation models like COCOMO II [Betz and Mäkiö, 2007]. In future, the team characteristics and the communication distance can be also estimated using similar scales as those used in COCOMO II.

In addition to team structures, the estimation information (e.g. effort of components, precedence relationships) used in the planning experiments may bias the results. The estimation may vary from one project manager to another and may result in different work distribution proposals. However, for a given component information and team characteristics available for developing the work, the produced work distribution will be optimal as defined by the fitness function. Also, the applied decoupling solutions are fairly limited and each company may have pre-defined decoupling solutions in place, which limits the applicability of the approach. Although using more decoupling solutions would bring more value to the approach, the initial experiments show the applicability of the genetic algorithm (discussed in Section 4.2) in choosing suitable decoupling solutions for the proposed work distribution and schedule plans.

10 CONCLUSIONS

10.1 Thesis Questions Revisited

RQ-1: How to apply genetic algorithms for developing automated support for software architecture design and for planning GSD projects?

In order to apply genetic architecture synthesis, the initial design (representing functional decomposition and operation characteristics of the system) and quality requirements (weights of sub-fitnesses) of the target system need to be given as input. The genetic algorithm inserts and removes patterns to transform the initial design into an architectural proposal satisfying the given quality requirements. The fitness function is used for evaluating the quality of design. As per the experiment reported in Chapter 3, in many cases, the genetic algorithm has introduced such patterns that a human architect would have selected for the given quality requirements.

In the case of constraint satisfaction and optimization approach, the problem of applying design patterns and architecture styles to the initial design of a system is modeled as a constraint satisfaction and optimization problem and is solved using readymade constraint solvers. It has the capability to reason about why a particular decision is chosen. This kind of support would enable a decision maker to understand the rationale behind the chosen decision and facilitates the reuse of design experience.

The set of components to be developed and the set of teams available for developing them are given as input to the genetic algorithm approach for planning projects. Based on the input, a random work assignment and schedule is created. The genetic algorithm improves the work assignment and schedule by changing the team assigned to the component, changing the schedule in which components are developed and introducing or removing decoupling solutions between the components. The fitness function evaluates the work distribution and schedule plan with respect to cost and time required for developing the software. As per the experiments reported in Section 4.2, the genetic algorithm has favored low cost teams in response to decreasing the cost of the project. This kind of work allocation seems appropriate, as moving work to low cost teams would reduce the cost of the project.

RQ-2: How can an end user (with little knowledge of search-based techniques) express the input required to apply genetic algorithms for software architecture design and planning projects? How to present the results to an end user?

To address this research question this thesis has introduced the Darwin tool. For using Darwin, the end user does not require deep knowledge about underlying search algorithms. All the required input for applying the genetic algorithm based software architecture design and project planning approaches can be expressed using the different views of Darwin (as discussed in Chapter 6). Also, the input parameters can be altered in the beginning or during the architecture design process and the impact can be

immediately observed. Different views (e.g. fitness graph, generations view, class diagram view) are developed for presenting the results to the end users. These views can be used for viewing and analyzing the results produced in the intermediate generations. They give an overview of how the input has been evolved from its initial stage.

RQ-3: How can a human decision maker and genetic algorithm-based automated support collaborate in solving software architecture design and project planning problems?

To answer this research question, two collaborative methods were introduced in this thesis. The first method introduces a semi-automated architecture design process, where the architect can decide which parts of the architecture proposals produced by the automated support are not optimal, freeze them, and can make other possible modifications to the architecture and can resubmit the modified proposal to the automated support for more fine grained improvements. To realize such interactive collaboration, the Darwin tool has been extended with three new mechanisms: allowing the existing architectures to be used as input, allowing the freezing of certain parts of the input architecture, and allowing marking unwanted patterns in the architecture. As per the empirical study reported in Subsection 7.1.3, the semi-automated architecture design approach is able to produce architecture proposals comparable to those designed by senior software engineering students.

The second method presents the decision maker with a set of non-dominated solutions for deciding on the required solution. This kind of collaboration is possible through multi-objective fitness graph view (discussed in Section 7.2). This view presents a visual curve representing a set of non-dominated work distribution plans in cost and time space. The decision maker can then analyze these work distribution plans and can choose the one appropriate for the problem at hand.

RQ-4: How can genetic algorithm-based approaches be made more effective for the multi-objective software architecture design and project planning?

This thesis proposes two methods to cope with the multi-objective project planning and software architecture design problems. The first method explores the use of Pareto optimality for multi-objective planning in GSD projects. The Pareto optimal genetic algorithm evaluates each work distribution plan individually for two objectives, i.e., cost and duration and produces a set of optimal work distribution and schedule plans in cost and time space. As per the experiments reported in Section 4.3, the genetic algorithm has allocated majority of the work units that need communication to same teams. This kind of work allocation seems sensible, as moving dependent work units to same teams decrease the communication overhead between the teams. Moreover, the approach can be used for exploring the trade-off between cost and completion time of the project.

The second method introduces a quality farm approach to accelerate the genetic architecture synthesis towards optimal multi-objective solutions. The farm approach is based on separately developing individual farms, where fitness function optimizes individuals for only one objective. Next, the farms are crossbred to form a new population, combining the best parts of both farms. The genetic algorithm then evolves the obtained new population using a balanced fitness function, which rewards multiple objectives. The initial results (presented in Subsection 5.1.2) show that the use of farm-

based approach has accelerated the genetic architecture synthesis towards multiple objective solutions.

10.2 Future Work

The approaches for architecture synthesis could be extended to include new quality attributes of software architectures. Introducing new patterns and quality attributes to genetic architecture synthesis approach is a cumbersome task, as solutions and fitness function are hard-coded in the current setup. A new specification language for seamlessly adding solutions to the genetic algorithm and for easily tuning the fitness function would reduce the effort. One interesting research topic would be to construct a dynamically expanding solution repository, which is able to adopt new solutions when an architect makes a decision to apply a solution that is not yet in the repository. The explanations provided by the constraint solver are a bit difficult to interpret. An interactive editor which translates the explanations into a human readable text would be beneficial. Also, it would be interesting to apply constraint satisfaction techniques for studying the GSD work planning problem. Moreover, a further study on the formal comparison of constraint satisfaction based architecture design and genetic architecture synthesis approach needs to be performed.

An obvious future work in project planning is to consider previous project data for estimating the communication distance and effort of components. Also, it would be interesting to study different decoupling solutions that are used in practice and how they influence the team communication. Another extension could be to take the quality of the architecture into planning process. This would make the approach more practical, where the project manager tries to balance between scope of the project, cost and schedule of the project.

The proposed approaches need to be further investigated by applying them to design and plan new example systems, preferably from the industry. The obtained results need to be evaluated together with the industry experts and the approaches need to be further fine-tuned based on their feedback. Moreover, the evaluation functions used in the proposed approaches for measuring the quality of solutions need to be further extended to be applicable for the real-world systems.

One interesting research area would be to apply proposed approaches in the context of agile development practices. Agile development practices use iterative cycles to develop the software. Each iterative cycle includes requirement analysis, design, planning, evolution and delivery phases. It would be interesting to study how the proposed approaches could support agile development. For example, the proposed approaches can be used to come up with the architecture design and work plan proposals for every iteration cycle. The incremental architecture design approach discussed in Section 7.1 is already suitable for designing the architecture for each iterative cycle. However, the project planning approach needs to be extended to support agile development by considering the features to be released for each iterative cycle.

Furthermore, the proposed approaches could be applied for managing the technical debt of a system [Cunningham, 1999]. The technical debt is inevitable in software systems and if not managed properly it may lead to rewriting of software from scratch. Some work related to managing technical debt using genetic algorithms is presented in [Vathsavayi and Systä, 2016]. Also, it would be interesting to investigate the applicability of the proposed architecture design approaches to the domain of distributed

control systems, where a wide variety of domain specific patterns are already available [Eloranta et al., 2014].

10.3 Final Remarks

Software architecture design and project planning are non-trivial and demanding tasks for software engineers to perform. The constant increase in the complexity and size of software systems demands a continuous search for better methods for software architecture design and project planning. In this thesis, the potential of techniques like genetic algorithms and constraint satisfaction techniques is studied in developing methods for automated software architecture design and project planning.

The genetic architecture synthesis approach was able to produce good architectures in complex design situations with multiple quality attributes. The approach was unbiased in choosing the solutions and if equipped with sufficient set of patterns, the approach may suggest fresh solutions that an architect restricted by her previous experience would not even think of. This kind of architecture synthesis approach would be beneficial in the agile development approaches, where much effort is not spent on designing the architecture of the system. As a consequence of lack of proper architecture design, unexpected rework costs are required in the later stages of development. This can increase the cost of the project and can also reduce the time-to-market. If an automated architecture synthesis approach is used in the beginning of each iteration cycle to come up with a solid architecture, then unexpected rework costs can be avoided in the later iterations.

In the context of GSD, the different team attributes and challenges that have to be taken into account in assigning work to teams located in distributed sites could be a very challenging task for the project manager. It is sensible to use automated support for such a complex multi-objective task. The automated support can also choose a solution without having bias towards a specific solution. Moreover, it would be beneficial to integrate this kind of approach with the software configuration management systems used in the organizations. In that case, the data generated from the software configuration management systems can be automatically processed to collect information about the team characteristics and relationships. Based on that information much more meaningful work distributions can be proposed and also meaningful predictions can be made about future projects. Also, the generated data can be analyzed for deducing the actual process followed by the teams in developing the project. This information can be used to know possible gaps in the adopted software process and can be further used for improving the development process.

The developed tool support Darwin eases the user in applying the proposed genetic algorithm approaches for software architecture design and project planning. It also provides much more information on the development of solutions to the user. Through the interaction support provided by Darwin the architect can use her expertise in guiding the genetic algorithm towards the optimal solutions. Moreover, Darwin provides guidance for the project manager in choosing a suitable work distribution plan for the available project constraints. This kind of support would be helpful for the manager to perform what-if analysis and choose a suitable work distribution for the project at hand and for studying possible advantages and disadvantages obtained from distributing the work to different sites.

The author is optimistic that over the next few years the proposed approaches will be spread to industrial practice. The target is challenging, but not completely unreachable. In the current stage, the proposed approaches can be seen as recommendation systems to come up with an initial architecture or work distribution proposal, which the architect or project manager can later modify into a more refined proposal using their expertise. These techniques significantly ease the workload of the architects or project managers in coming up with a good design and work distribution proposal.

REFERENCES

- [Alba et al., 2007] E. Alba, F. Chicano, Software Project Management with GAs. *Information Sciences*, **177** (11), 2007, 2380-2401.
- [Albert and Dieter Rombach, 2003] E. Albert and H. Dieter Rombach, *A handbook of software and systems engineering: empirical observations, laws, and theories*. Addison Wesley, 2003.
- [Aleti et al., 2009] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya, ArcheOptrix: an extendable tool for architecture optimization of AADL models. In: *Proceedings of ICSE'09 Workshop MOMPES'09*, 2009, 61-71.
- [Aleti et al., 2013] A. Aleti, B. Buhnova, L. Grunske, A. Koziolk, and I. Meedeniya, Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, **39** (5), 2013, 658-683.
- [Ali et al., 2010] N. Ali, S. Beecham and I. Mistrík, Architectural Knowledge Management in Global Software Development: A Review. In: *Proceedings of the International Conference on Global Software Engineering (ICGSE'10)*, 2010, 347-352.
- [Aljawawdeh et al., 2015] H. Aljawawdeh, C. Simons and M. Odeh, Metaheuristic design pattern: Preference. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, 2015, 1257-1260.
- [Allen, 1977] T. J. Allen, *Managing the Flow of Technology*. MIT Press, 1977.
- [Almeida et al., 2011] L.H. Almeida, P.R. Pinheiro, and A.B. Albuquerque, Applying Multi-Criteria Decision Analysis to Global Software Development with Scrum Project Planning. In: *Proceedings of RSKT'11*, LNCS 6954, Springer, 2011, 311-320.
- [Amoui et al., 2006] M. Amoui, S. Mirarab, S. Ansari and C. Lucas, A genetic algorithm approach to design evolution using design pattern transformation. *International Journal of Information Technology and Intelligent Computing*, **1** (1, 2), 2006, 235-245.
- [Antoniol et al., 2011] G. Antoniol, M. Di Penta, M. Harman, The Use of Search-based Optimization Techniques to Schedule and Staff Software Projects: An Approach and an Empirical Study. *Software Practice and Experience*, **41** (5), 2011, 495-519.

- [Arcuri et al., 2011] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: *Proceedings of ICSE'11*, 2011, 1-10.
- [Arvritzer et al., 2008] A. Arvritzer, D. Paulish, C. Yuanfang, Coordination implications of software architecture in a global software development project. In: *Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 2008)*, 2008, 107-116.
- [Avgeriou et al., 2007] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham and P. Dewayne, Architectural Knowledge and Rationale - Issues, Trends, Challenges. *ACM Sigsoft Software Engineering Notes*, **32** (4), 2007, 41-46.
- [Barreto et al., 2008] A. Barreto, M de O. Barros, C. M. L. Werner, Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers and Operations Research*, **35** (10), 2008, 3073-3089.
- [Barros and Dias-Neto, 2011] M. O. Barros, A. C. Dias-Neto, *Threats to Validity in Search-based Software Engineering Empirical Studies*, Technical Report DIA/UNIRIO, No. 6, Rio de Janeiro, Brazil, 2011. (available at the following URL <http://www.seer.unirio.br/index.php/monografiasppgi/article/viewFile/1479/1307>)
- [Bartrak, 1999] R. Barták, Constraint Programming: In Pursuit of the Holy Grail, In: *Proceedings of the Week of Doctoral Students (WDS 99)*, Part IV, 1999, 555-564.
- [Bass et al., 1998] L. Bass, P. Clements and R. Kazman, *Software architecture in practice*. Addison-Wesley, 1998.
- [Bass and Paulish, 2004] M. Bass and D. Paulish, Global Software Development Process Research at Siemens. In: *Proceedings of the third International Workshop on Global Software Development*, 2004, 8-11.
- [Betz and Mäkiö, 2007] S. Betz, J. Mäkiö, Amplification of the COCOMO II regarding offshore software projects, In: *Workshop on Offshoring of Software Development- Methods and Tools for Risk Management at the second International Conference on Global Software Engineering*, 2007, 33-46.
- [Boehm, 2000] B. Boehm, C. Abts, A. Brown, S. Chulani, *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.
- [Bosch, 2000] J. Bosch, *Design and Use of Software Architectures – Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.
- [Bowman et al., 2010] M. Bowman, L.C. Briand, Y. Labiche, Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. *IEEE Transactions on Software Engineering*, **36** (6), 2010, 817–837.
- [Britto et al., R. Britto, P. S. Neto, R. Rabelo, W. Ayala, and T. Soares, A Hybrid

- 2012] Approach to Solve the Agile Team Allocation Problem. In: *Proceedings of the Conference on Evolutionary Computation (CEC'12)*, Brisbane, Australia, 2012, pp. 1-8.
- [Brown et al., 1998] W.J. Brown, R.C. Malveau, H.W. McCormickIII and T.J. Mowbray, *Antipatterns - Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1998.
- [Burgess, 1995] C.J. Burgess, A Genetic algorithm for the optimisation of a multiprocessor computer architecture. In: *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95)*, 1995, 39-44.
- [Burke et al., 2004] E.K. Burke, S. Gustafson, and G. Kendall, Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, **8** (1), 2004, 47-62.
- [Buschmann et al., 1996] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley, 1996.
- [Caramel and Tjia, 2005] E. Carmel and P. Tjia, *Outsourcing Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, 2005.
- [Castro et al., 2008] C.B. Castro, H. Astudillo, Weak Constraint Programming to Identify Alternative Composite COTS-Based Software Systems from Imperfect Information, In: *Proceedings of the International Conference of the Chilean Computer Science Society (SCCC '08)*, 2008, 62-69.
- [Chang et al., 2001] C.K. Chang, M.J. Christensen, T. Zhang, Genetic Algorithms for Project Management. *Annals of Software Engineering*, **11**, 2001, 107-139.
- [Chicano et al., 2011] F. Chicano, F. Luna, A. J. Nebro, and E. Alba, Using Multiobjective Metaheuristics to Solve the Software Project Scheduling Problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'11)*, 2011, 1915-1922.
- [Chidamber and Kemerer, 1994] S.R. Chidamber and C.F. Kemerer, A metrics suite for object oriented design, *IEEE Transactions on Software Engineering*, **20** (6), 1994, 476-492.
- [Clerc et al., 2005] V. Clerc, P. Lago, and H. van Vliet, Global Software Development: Are Architectural Rules the Answer? In: *Proceedings of ICGSE'07*, 2007, 225-234.
- [Cochran et al., 2003] J.K. Cochran, S.M. Horng, and J.W. Fowler, A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers and Operations Research*, **30** (7), 2003, 1087-1102.

- [Conway, 1968] http://www.melconway.com/Home/Conways_Law.html, Checked January 2015.
- [Costa et al., 2010] C. Costa, C. Cunha, R. Rocha, C. França, F. Silva and R. Prikladnicki, Models and tools for managing distributed software development: A systematic literature review. In: *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2010, 73-76.
- [Cunningham, 1999] W. Cunningham, The wycash portfolio management system. In: *Proceedings of the 7th annual conference on Object-Oriented Programming Systems Languages and Applications*, 1992, 29-30.
- [Darwin tool, 2016] <http://practise.cs.tut.fi/project.php?project=darwin&page=downloads>, Checked September 2016.
- [de Boer and Farenhorst, 2008] R.C. de Boer, R. Farenhorst, In search of architectural knowledge. In: *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge*, 2008, 71–78.
- [Deb, 1999] K. Deb, Evolutionary algorithms for multicriterion optimization in engineering design. In: *Proceedings of EUROGEN'99*, 1999, 135-161.
- [Debels et al., 2005] D. Debels, M. Vanhoucke, A Bi-population Based Genetic Algorithm for the Resource-Constrained Project Scheduling Problem. In: *Proceedings of ICCSA'05*, 2005, 378-387.
- [DeLone et al., 2005] W. Delone, J. A. Espinosa, G. Lee and E. Carmel, Bridging Global Boundaries for IS Project Success. In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, 1-8.
- [Di Penta et al., 2005] M. Di Penta, M. Neteler, G. Antoniol and E. Merlo, A language independent software renovation framework. *The Journal of Systems and Software*, **77** (3), 2005, 225-240.
- [Dolado and Fernandez, 1998] J. J. Dolado, L. Fernandez, Genetic Programming, Neural Networks and Linear Regression in Software Project Estimation. In: *Proceedings of International Conference on Software Process Improvement*, 1998, 157–171.
- [Dolado, 2001] J. J. Dolado, On the Problem of the Software Cost Function. *Information and Software Technology*, **43**, 2001, 61-72.
- [Duggan et al., 2004] J. Duggan, J. Byrne and G.J. Lyons, A Task Allocation Optimizer for Software Construction. *IEEE Software*, **21** (3), 2004, 76-82.
- [Eclipse, 2015] <http://www.eclipse.org>, checked March 2015.
- [El-Boussaidi and Mili, 2008] G. El-Boussaidi, H. Mili, Detecting Patterns of Poor Design Solutions Using Constraint Propagation. In: *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, 2008,

- [Eloranta et al., 2014] V. Eloranta, J. Koskinen, M. Leppänen and V. Reijonen, *Designing Distributed Control Systems – A Pattern Language approach*. John Wiley & Sons, 2014.
- [Etemaadi et al., 2012] R. Etemaadi, M. Emmerich, and M. Chaudron, Problem-specific search operators for metaheuristic software architecture design. In: *Proceedings of 4th Symposium on Search Based Software Engineering*, 2012, 267-272.
- [Fernandez et al., 2012] J. Fernandez and M. Basavaraju, Task Allocation Model in Globally Distributed Software Projects Using Genetic Algorithms. In: *Proceedings of the Seventh International Conference on Global Software Engineering (ICGSE'12)*, 2012, 181.
- [Ferucci et al., 2010] F. Ferrucci, C. Gravino, R. Oliveto and F. Sarro, Using Evolutionary Based Approaches to Estimate Software Development Effort. In: Chis M (ed) *Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques*, IGI Global, 2010, 13-28.
- [Ferrucci et al., 2013] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, Not Going to Take this Anymore: Multi-Objective Overtime Planning for Software Engineering Projects. In: *Proceedings of the 35th International Conference on Software Engineering*, 2013, 462-471.
- [Ferrucci et al., 2014] F. Ferrucci, M. Harman, and F. Sarro. Search based software project management. *Software Project Management in a Changing World*, 2014, 373-399.
- [Friedman, 2007] T. Friedman, *The world is flat*. Picador, 2007.
- [Gamma et al., 1995] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [Garlan et al., 2001] D. Garlan, S. W. Cheng, A.C. Huang, B. Schmerl and P. Steenkiste, Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, **37** (10), 2004, 46-54.
- [Gen et al., 2008] M. Gen, R. Cheng, L. Lin, *Network models and optimization: Multiobjective genetic algorithm approach*. Springer, 2008.
- [Ghosh and Dehuri., 2004] A. Ghosh, S. Dehuri, Evolutionary algorithms for multi-criterion optimisation: a survey. *International Journal of Computing & Information Sciences*, **2** (1), 2004, 38–57.
- [Glavaš and Fertalj, 2011] G. Glavaš, K. Fertalj, Solving the class responsibility assignment problem using metaheuristic approach. *Journal of Computer and Information technology*, **19** (4), 2011, 275–283.
- [Guennecc et al., 2000] A.L. Guennecc, G. Sunye, and J. Jezequel, Precise Modeling of Design Patterns. In: *Proceedings of International conference on Unified*

Modeling Language, 2000, 482-496.

- [Gueheneuc et al., 2001] Y-G. Gueheneuc, H. Albin-Amiot, Using design patterns and constraints to automate the detection and correction of inter-class design defects. In: *Proceedings of 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems*, 2001, 296-305.
- [Harman and Jones, 2001] M. Harman and B. F. Jones, Search based software engineering. *Information and Software Technology*, **43** (14), 2001, 833-839.
- [Harman, 2007] M. Harman, The Current State and Future of Search Based Software Engineering. In: *Proceedings of Future of Software engineering (FOSE '07)*, 2007, 342-357.
- [Harman, 2012] M. Harman, The role of Artificial Intelligence in Software Engineering. In: *Proceedings of the First International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 2012, 1- 6.
- [Harman et al., 2012] M. Harman, P. McMinn, J. Souza, and S. Yoo, Search based software engineering: Techniques, taxonomy, tutorial. In B. Meyer and M. Nordio, editors, *Empirical software engineering and verification: LASER 2009-2010*, 2012, 1-59.
- [Harrison and Avgeriou, 2011] N. Harrison, P. Avgeriou, Pattern-Based Architecture Reviews. *IEEE Software*, **28** (6), 2011, 66-71.
- [Hevner et al., 2004] A. R. Hevner, S. T. March, J. Park, and S. Ram, Design Science in Information Systems research. *MIS Quarterly*, **28** (1), 2004, 75-105.
- [Herbsleb and Grinter, 1999] J.D. Herbsleb and R.E. Grinter, Splitting the Organization and Integrating the Code: Conway's Law Revisited. In: *Proceedings of International Conference on Software Engineering*, 1999, 85-95.
- [Herbsleb and Mockus, 2003] J.D. Herbsleb, A. Mockus, An Empirical Study of Speed and Communication in Globally-Distributed Software Development. *IEEE Transactions on Software Engineering*, 29 (6), 2003, 125-134.
- [Hohpe et al., 2004] G. Hohpe and B. Woolf, *Enterprise Integration Patterns*. Addison-Wesley, 2004.
- [Holland, 1975] J.H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, Ann Arbor, 1975.
- [Hossain et al., 2011] E. Hossain, P. L. Bannerman, and D. R. Jeffery, Scrum practices in global software development: a research framework. In: *Proceedings of the 12th International Conference on Product-focused Software Process Improvement*, 2011, 88–102.

- [Husbands, 1989] P. Husbands, Genetic algorithms for scheduling. *AISB Quarterly*, (89).
- [IEEE, 2011] IEEE Recommended Practice for Architectural Description of Software Intensive Systems. *IEEE Standard 42010-2011*, 2011.
- [Jansen and Bosch, 2005] A.G.J. Jansen, J. Bosch, Software Architecture as a Set of Architectural Design Decisions. In: *Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 2005)*, 2005, 109-119.
- [Jansen, 2008] A.G.J. Jansen, Architectural Design Decisions. Ph.D. thesis, University of Groningen, Netherlands, 2008. <http://irs.ub.rug.nl/ppn/314295305> checked February 2015.
- [JFreeChart, 2015] JFreeChart, <http://www.jfree.org/>, visited in February 2015.
- [Jira, 2015] <https://www.atlassian.com/software/jira>, checked April 2015.
- [Jones et al., 1998] B.F. Jones, D.E. Eyres, H.H. Sthamer, A Strategy for using genetic algorithms to automate branch and fault-based testing. *The Computer Journal*, 41(2), 1998, 98-107.
- [Julstrom, 1994] B.A. Julstrom, Seeding the population: improved performance in a genetic algorithm for the rectilinear Steiner problem. In: *Proceedings of the ACM Symposium on Applied Computing (SAC'94)*, 1994, 222-226.
- [Jussien and Barichard, 2000] N. Jussien and V. Barichard, The PaLM system: explanation-based constraint programming. In: *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems*, a post-conference workshop of CP 2000, 2000, 118-133.
- [Jussien, 2005] N. Jussien, Constraint programming for software engineering. In: *Proceedings of the Fifth Congress of Logic applied to Technology (LAPTEC'05)*, 2005.
- [Keil et al., 2006] P. Keil, D. J. Paulish, R. Sangwan, Cost estimation for global software development. In: *Proc. International Workshop on Economics Driven Software Engineering*, 2006, 7-10.
- [Kim et al., 2003] D. Kim, R. France, S. Ghosh, and E. Song, A Role-Based Metamodeling Approach to Specifying Design Patterns. In: *Proceedings of International Computer Software and Applications (COMPSAC)*, 2003, 452-457.
- [Kitchin, 1994] Robert M. Kitchin, Cognitive maps: what are they and why study them? *Journal of Environmental Psychology*, **14** (1), 1994, 1 -19.
- [Klau et al., 2010] G. Klau, N. Lesh, J. Marks and M. Mitzenmacher, Human-guided search. *Journal of Heuristics*, **16** (3), 2010, 289–310.

- [Kramer and Magee, 1985] J. Kramer and J. Magee, Dynamic configuration for distributed systems. *IEEE Transactions on Software Engineering*, **11** (4), 1985, 424-436.
- [Krasner and Pope, 1988] Glenn E. Krasner , Stephen T. Pope, A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, **1** (3), 1988, 26-49.
- [Lamersdorf and Münch, 2010] A. Lamersdorf, A. and J. Münch, A multi-criteria distribution model for global software development projects. *Journal of the Brazilian Computer Society*, **16** (2), 2010, 97-115.
- [Lamersdorf and Munch, 2009] A. Lamersdorf and J. Munch, Tamri: A tool for supporting task distribution in global software development projects. In: *Proceedings of ICGSE'09*, 2009, 322–327.
- [Lamersdorf , 2011] A. Lamersdorf. Model-based decision support of task allocation in global software development. Ph.D. thesis, 2011. <http://publica.fraunhofer.de/eprints/urn:nbn:de:0011-n-796090.pdf> checked January 2015.
- [Li et al., 2011] R. Li, R. Etemaadi, M.T.M. Emmerich, and M.R.V. Chaudron, An Evolutionary Multiobjective Optimization Approach to Component-Based Software Architecture design. In: *Proceedings of the Conference on Evolutionary Computation (CEC'11)* 2011, 432-439.
- [Lundberg et al., 1999] L. Lundberg, J. Bosch, D. Häggander, and P.O. Bengtsson, Quality attributes in software architecture design. In: *Proceedings of the Third International Conference on Software Engineering and Applications*, 1999, 353-362.
- [Mak and Kruchten, 2007] D.K.M. Mak, P.B. Kruchten, NextMove: A Framework for Distributed Task Coordination. In: *Proceedings of Australian Software Engineering Conference (ASWEC'07)*, 2007, 399-408.
- [Mancoridis et al., 1998] S. Mancoridis, B.S. Mitchell, C. Rorres, Y.F. Chen and E.R. Gansner, Using automatic clustering to produce high-level system organizations of source code. In: *Proceedings of International Workshop on Program Comprehension (IWPC 98)*, 1998, 45-53.
- [Martens et al., 2010] A. Martens, H. Koziolk, S. Becker and R. Reussner, Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms. In: *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, 2010, 105-116.
- [McGregor et al., 2007] J. McGregor, F. Bachmann, I. Bass and P. Bianco, using ArchE in the classroom: one experience. Technical Note, CMU/SEI-2007-TN001, 2007.
- [MDTtools, 2015] <http://eclipse.org/modeling/mdt/?project=uml2tools>, checked March 2015.

- [Michalewicz, 1992] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionary Programs*. Springer-Verlag, 1992.
- [Michalewicz, 1995] Michalewicz Z. A survey of constraint handling techniques in evolutionary computation methods. In: *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, 1995, 135-155.
- [Microsoft Project, 2015] <https://products.office.com/en-us/project/>, checked April 2015
- [Minku et al., 2012] L.L. Minku, D. Sudholt, X. Yao, Evolutionary Algorithms for the Project Scheduling Problem: Runtime Analysis and Improved Design. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'12)*, 2012, 1221-1228.
- [Mitchell, 1996] M. Mitchell, *An introduction to Genetic Algorithms*. MIT Press, 1996.
- [Mitchell et al., 2002] B.S. Mitchell, S. Mancoridis and M. Traverso, Search based reverse engineering. In: *Proceedings of 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, 2002, 431-438.
- [Mockus and Weiss, 2001] A. Mockus and David M. Weiss, Globalization by chunking: a quantitative approach. *IEEE Software*, 18 (2), 2001, 30-37.
- [Murata et al., 1996] T. Murata, H. Ishibuchi, H. Tanaka, Multi-objective genetic algorithm and its application to flowshop scheduling. *Computers and Industrial Engineering*, **30** (4), 1996, 957-968.
- [O'Keeffe and Ó Cinnéide, 2004] M. O'Keeffe and M. Ó Cinnéide, "Towards automated design improvements through combinatorial optimization," In: *Proceedings of 4th International Workshop on Directions in Software Engineering Environments (WoDiSEE2004)*, 2004, 75 – 82.
- [O'Keeffe and Ó Cinnéide, 2006] M. O'Keeffe and M. Ó Cinnéide, Search-based software maintenance. In: *Proc. Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006, 249-260.
- [O'Keeffe and Ó Cinnéide, 2008] M. O'Keeffe and M. Ó Cinnéide, Search-based refactoring for software maintenance. *Journal of Systems and Software*, 81 (4), 2008, 502-516.
- [Obitko, 2015] <http://www.obitko.com/tutorials/genetic-algorithms/>, checked in February 2015.
- [OpenProject, 2015] <https://www.openproject.org/>, checked April 2015.
- [Peixoto et al., 2014] D.C.C. Peixoto, G.R. Mateus, R.F. Resende, The Issues of Solving Staffing and Scheduling Problems in Software Development Projects. In: *Proceedings of the 38th Annual Computer Software and Applications*

Conference (COMPSAC), 2014, 1-10.

- [Rahnamayn et al., 2007] S. Rahnamayn, H. R. Tizhoosh, and S. Salama, A Novel Population Initialization Method for Accelerating Evolutionary Algorithms. *Computers and Mathematics with Applications*, **53** (10), 2007, 1605-1614.
- [Räihä et al., 2008] O. Räihä, K. Koskimies, E. Mäkinen, Genetic Synthesis of Software Architecture. In: *Proceedings of International Conference on Simulated Evolution and Learning*, 2008. 565-574.
- [Räihä et al., 2009] O. Räihä, K. Koskimies, E. Mäkinen, Scenario-Based Genetic Synthesis of Software Architecture. In: *Proceedings of ICSEA'09*, 2009, 437-445.
- [Räihä, 2010] O. Räihä, A survey on search-based software design. *Computer Science Review*, **4** (4), 2010, 203-249.
- [Räihä et al., 2010] O. Räihä, K. Koskimies and E. Mäkinen, Complementary Crossover for Genetic Software Architecture Synthesis. In: *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA'10)*, 2010, 260-265.
- [Räihä et al., 2011] O. Räihä, K. Koskimies, E. Mäkinen, Generating Software Architecture Spectrum with Multi-Objective Genetic Algorithms. In: *Proceedings of the Third World Congress on Nature and Biologically Inspired Computing (NaBIC'11)*, 2011, 29-36.
- [Riehle, 2000] D. Riehle. Framework design: A role modeling approach. PhD Thesis, Swiss Federal Institute of Technology Zurich, Universitat Hamburg, 2000. <http://dirkriehle.com/computer-science/research/dissertation/> checked January 2015.
- [Roy and Veraart, 1996] G. G. Roy, V.E. Veraart, Software Engineering Education: from an Engineering Perspective, In: *Proceedings of the 1996 International Conference on Software Engineering: Education and Practice*, 1996, 256-263.
- [Salger, 2009] F. Salger, Software architecture evaluation in global software development projects. In: *Proceedings of the On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, 2009, 391-400.
- [Sarro et al., 2012] F. Sarro, F. Ferrucci, and C. Gravino, Single and Multi Objective Genetic Programming for Software Development Effort Estimation. In: *Proceedings of 27th Annual ACM Symposium on Applied Computing*, 2012, 1221-1226.
- [Sayyad et al., 2013] A.S. Sayyad, and H. Ammar, Pareto-Optimal Search-Based Software Engineering (POSBSE): A Literature Survey. In: *Proceedings of RAISE'13*, 2013, 21-27.
- [Schaffer, 1985] J. D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms. In: *Proceedings of First International Conference on*

Genetic Algorithms, 1985, 93–100.

- [Selonen et al., 2001a] P. Selonen P, T. Systä, K. Koskimies, Generating Structured Implementation Schemes from UML Sequence Diagrams. In: *Proceedings of Technology of Object-Oriented Languages and Systems*, 2001, 317-328.
- [Selonen et al., 2001b] P. Selonen, K. Koskimies and M. Sakkinen, How to make apples from oranges in UML. In: *Proceedings of the 34th Hawaii International Conference on System Sciences*, 2001, 1-10.
- [Seng et al., 2005] O. Seng, M. Bauyer, M. Biehl, G. Pache, Search-based improvement of subsystem decomposition. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'05*, 2005, 1045–1051.
- [Seng et al., 2006] O. Seng, J. Stammel, D. Burkhart, Search-based determination of refactorings for improving the class structure of object-oriented systems. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'06*, 2006, 1909–1916.
- [Seshagiri, 2006] G. Seshagiri, GSD: Not a business necessity, but a march of folly. *IEEE Software*, **23** (5), 2006, 63-64.
- [Setamanit et al., 2007] S. Setamanit, W. Wakeland, D. Raffo, Using simulation to evaluate global software development task allocation strategies. *Software Process: Improvement and Practice*, **12** (5), 2007, 491-503.
- [Shaw and Garlan, 1996] M. Shaw and D. Garlan, *Software Architecture- Perspectives of an Emerging Discipline*. Prentice Hall, 1996.
- [Shukla, 2000] K.K. Shukla, Neuro-Genetic Prediction of Software Development Effort. *Information and Software Technology*, **42** (10), 2000, 701–713.
- [Simons et al., 2010] C. L. Simons, I. C. Parmee, R. Gwynllyw, Interactive, evolutionary search in upstream object-oriented class design. *IEEE Transactions on Software Engineering*, **36** (6), 2010, 798–816.
- [Simons and Parmee, 2012] C. L. Simons and I. Parmee, Elegant, object-oriented software design via interactive, evolutionary computation. *IEEE Transactions on Systems, Man, and Cybernetics: Part C - Applications and Reviews*, **42** (6), 2012, 1797-1805.
- [Simons and Smith, 2013] C. Simons and J. Smith, A comparison of meta-heuristic search for interactive software design. *Soft Computing*, **17** (11), 2013, 2147-2162.
- [Simons et al., 2014] C. Simons, J. Smith, and P. White, Interactive ant colony optimization (iACO) in early lifecycle software design. *Swarm Intelligence*, **8** (2), 2014, 139-157.
- [Smite et al., 2010] D. Smite, C. Wohlin, T. Gorschek, R. Feldt, Empirical Evidence in Global Software Engineering: A Systematic Review. *Journal of*

Empirical Software Engineering, **15** (1), 2010, 91-118.

- [Smith and Simons, 2015] J. Smith and C. Simons, The influence of search components and problem characteristics in early life cycle class modelling. *Journal of Systems and Software*, 103, 2015, 440-451.
- [SOA, 2015] <http://www.soapatterns.org>, checked April 2015.
- [Stylianou and Andreou, 2013] C. Stylianou, A.S Andreou, A multi-objective genetic algorithm for intelligent software project scheduling and team staffing. *Intelligent Decision Technologies: An International Journal*, 7(1), 2013, 59-80.
- [Tsang, 1995] E. Tsang, *Foundations of Constraint Satisfaction*. Academic Press, 1995.
- [UML, 2015] <http://www.omg.org/technology/documents/formal/uml.htm>, checked March 2015.
- [van der Ven et al., 2006] J.S. van der Ven, A.G.J. Jansen, J.A.G. Nijhuis, and J. Bosch, *Rationale Management in Software Engineering*, chapter 16, 2006, 329-348.
- [Van Heesch and Avgeriou, 2011] U. Van Heesch and P. Avgeriou, Mature architecting –a survey about the reasoning process of professional architects. In: *Proceedings of the 9th IEEE/IFIP Working Conference on Software Architecture*, 2011, 260-269.
- [Vathsavayi and Systä, 2016] S. Vathsavayi and K. Systä, Technical Debt Management with Genetic Algorithms. In: *Proceedings of the 42nd Euromicro Conference on SEAA*, 2016, to appear.
- [Wegener et al., 1997] J.Wegener, H.H. Sthamer, B.F. Jones and D.E. Eyres, Testing real-time systems using genetic algorithms. *Software Quality Control*, **6** (2), 1997, 127-135.
- [Wohlin et al., 1999] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell and A. Wesslen, *Experimentation in Software Engineering: An Introduction*. 1999.
- [Yildiz et al., 2012] B.M. Yildiz, B. Tekinerdogan, and S. Cetin, A Tool Framework for Deriving the Application Architecture for Global Software Development Projects. In: *Proceedings of the International Conference on Global Software Engineering (ICGSE'12)*, 2012, 94-103.
- [Zelkowitz and Wallace, 1997] M.V. Zelkowitz and D. Wallace, Experimental Validation in Software Engineering. *Journal of Information and Software technology*, 39, 1997, 735-743.
- [Zest, 2015] <http://www.eclipse.org/gef/zest/>, checked March 2015.

- [Zitzler et al., 2000] E. Zitzler, K. Deb, and L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary Computation*, **8** (2), 2000, 173–195.
- [Ågerfalk et al., 2005] P. J. Ågerfalk, B. Fitzgerald, H. Holmström, B. Lings, B. Lundell, Ó. E. Conchúir, A framework for considering opportunities and threats in distributed software development. In: *Proceedings of the International Workshop on Distributed Software Development*, 2005, 47–61.

Appendices

Appendix A: Parameters used for Automatic Chocolate Vending Machine.....	109
Appendix B: Electronic Home Control System.....	110

Appendix A: Parameters used for Automatic Chocolate Vending Machine

Appendix A presents the mutation probabilities and fitness weights used for the automatic chocolate vending machine (ACVM) example system discussed in Section 3.2.

Table 1. Mutation probabilities used for ACVM

<i>Mutation</i>	<i>Probability</i>
Introduce message dispatcher	4
Remove message dispatcher	2
Create link to dispatcher	8
Remove link to dispatcher	4
Introduce server	7
Remove server	4
Introduce façade	6
Remove façade	1
Introduce mediator	7
Remove mediator	3
Introduce strategy	7
Remove strategy	1
Introduce Adapter	6
Remove Adapter	4
Introduce template method	7
Remove template method	4
Introduce interface	9
Remove interface	3
Null mutation	2
Crossover	11

Table 2. Fitness weights used for ACVM

<i>Sub-fitnesses</i>	<i>Weights</i>
Positive modifiability	50
Negative modifiability	15
Positive efficiency	5
Negative efficiency	5
Complexity	5

Appendix B: Electronic Home Control System

Appendix B describes the creation of initial design of electronic home control system (i.e., ehome), which is used as an example system in this thesis. The first step is to identify the relevant functional requirements of ehome. The major use cases of ehome are shown in Figure 1. The user can login to ehome system and can manage the home by changing the temperature, moving the drape position, making coffee and playing music. Next each use case is refined into sequence diagrams representing the interaction between major components required for fulfilling the use cases. The corresponding sequence diagrams are presented in Figures 2, 3, 4, 5 and 6.

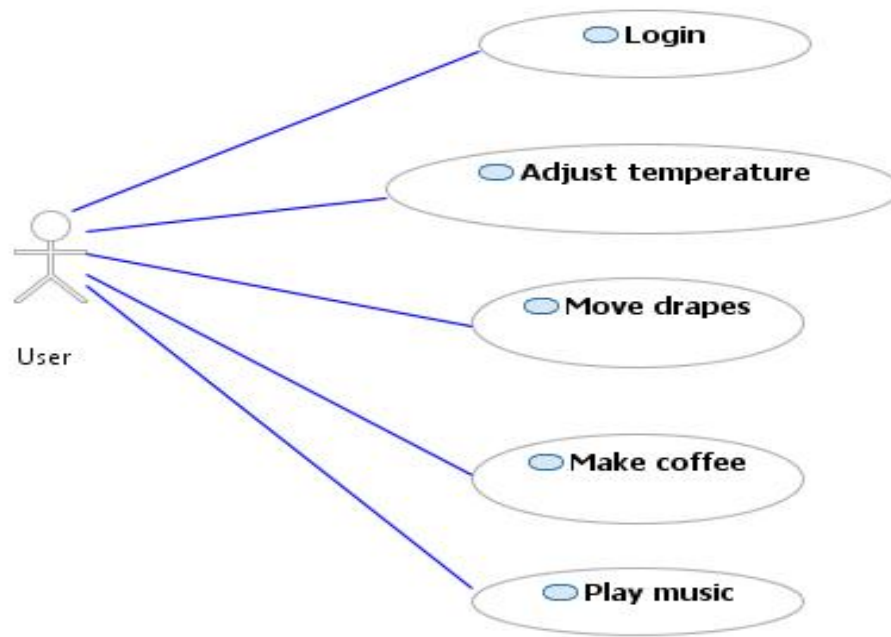


Figure 1. Abstract use cases of Ehome

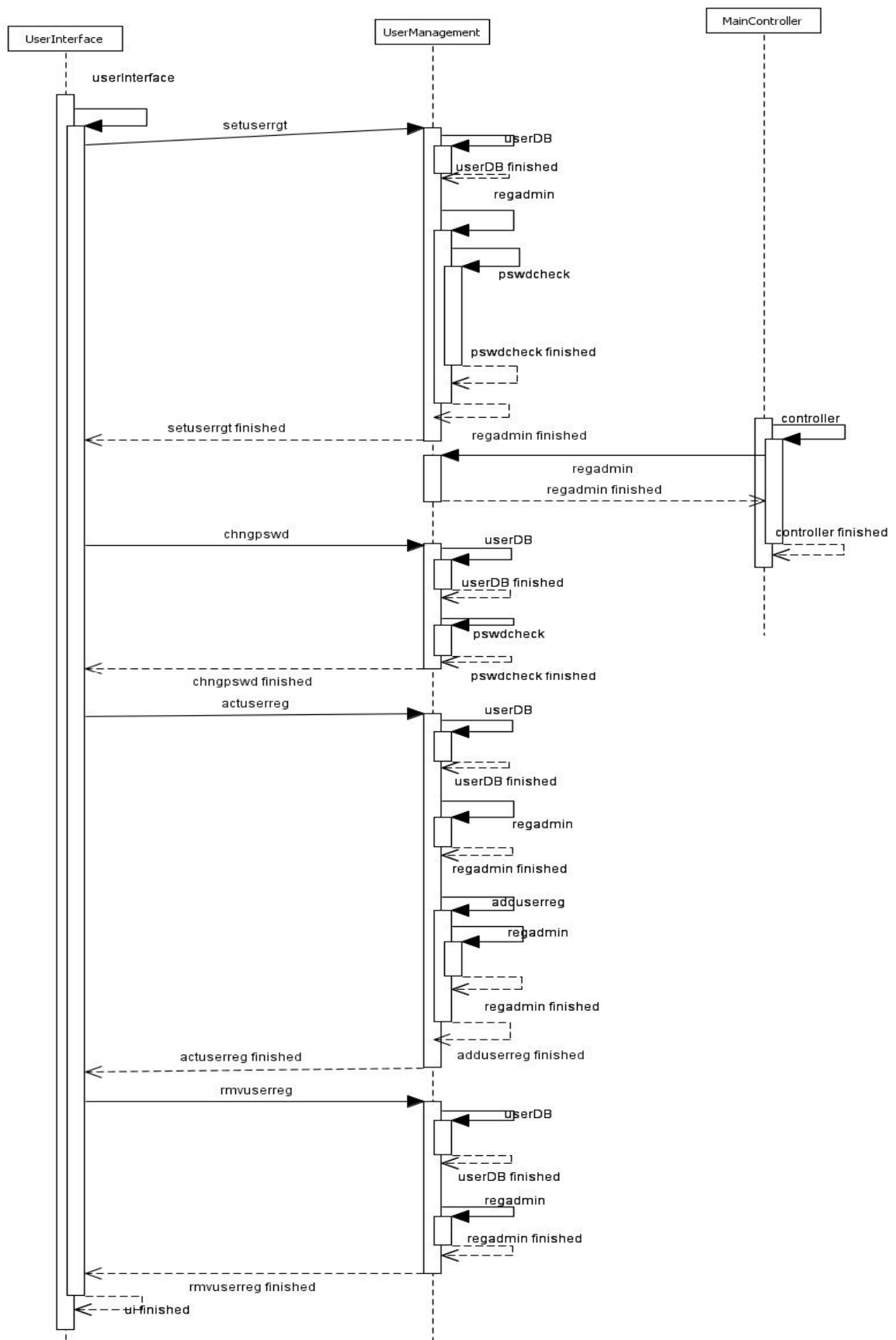


Figure 2. Sequence diagram for use case “login to ehome”

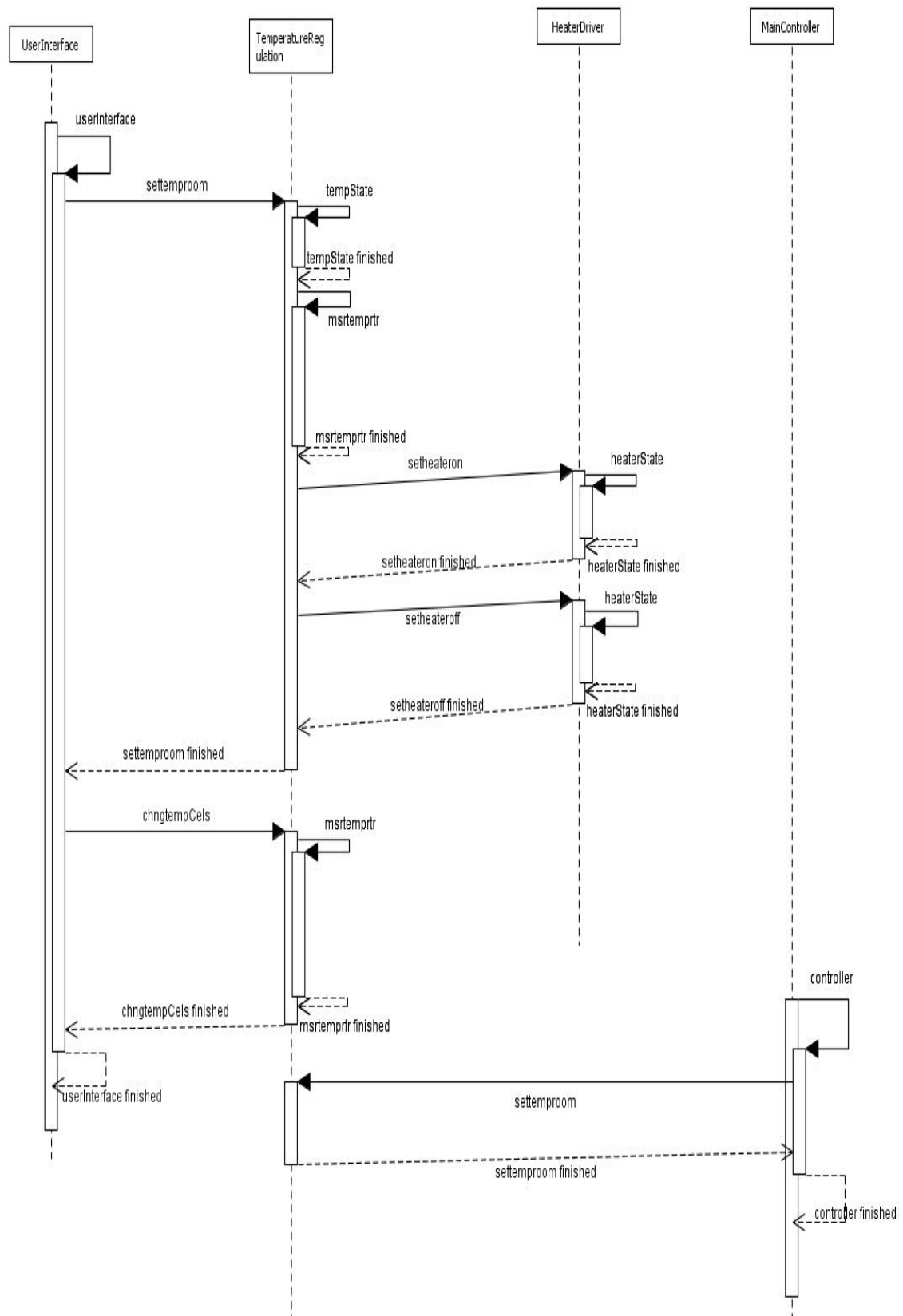


Figure 3. Sequence diagram for use case “adjust temperature”

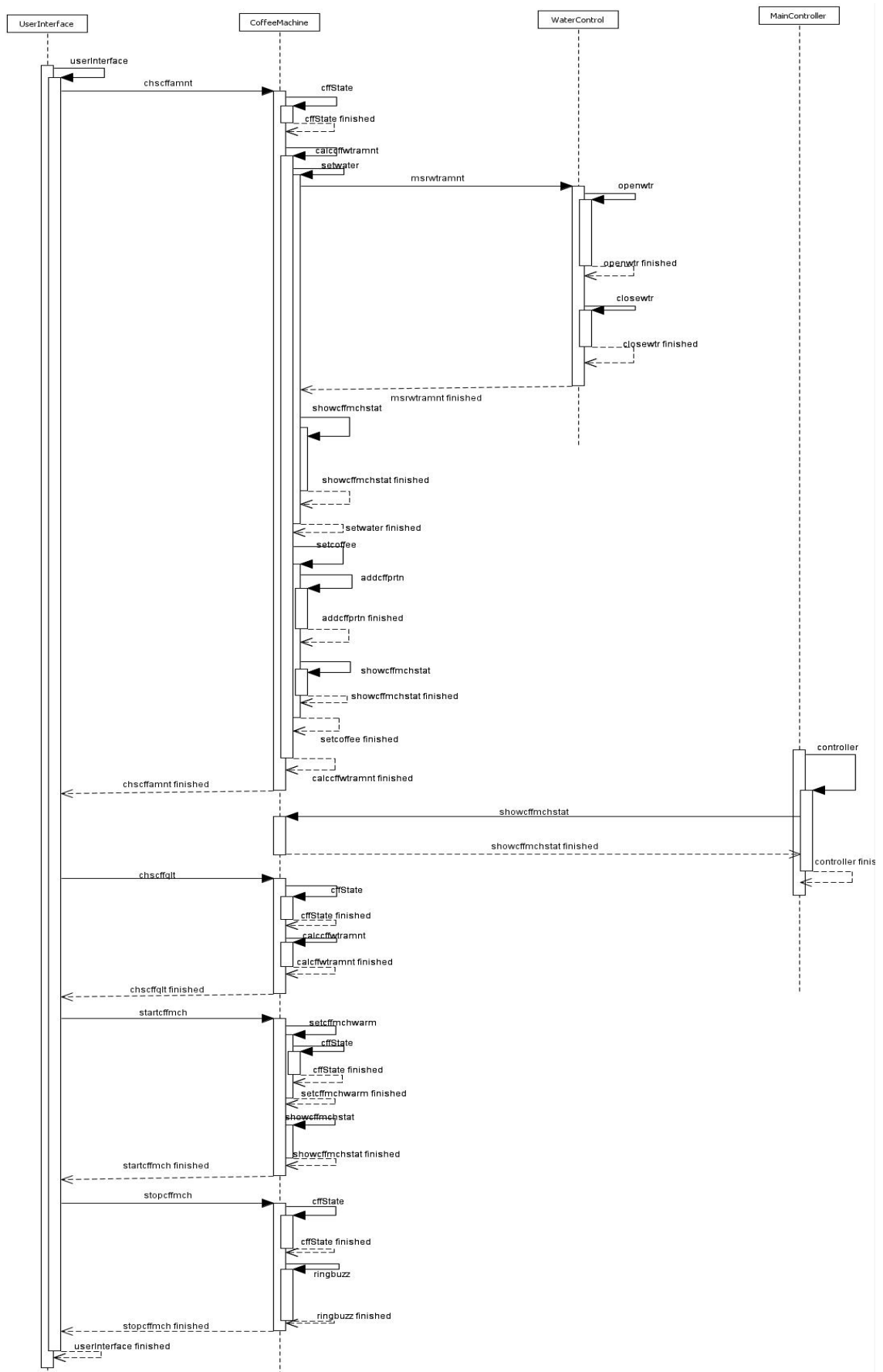


Figure 5. Sequence diagram for use case “make coffee”

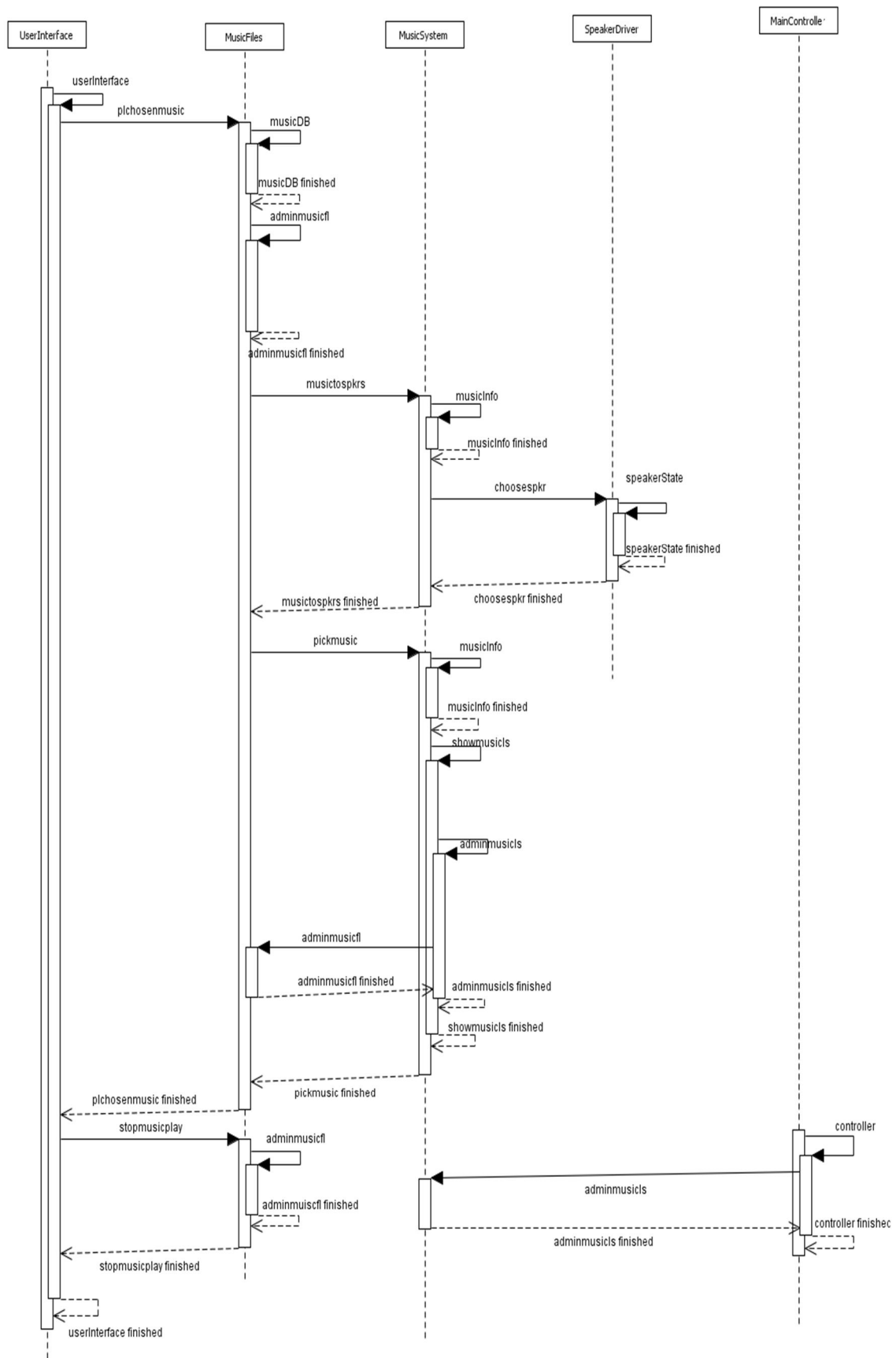


Figure 6. Sequence diagram for use case “play music”

The participants in the sequence diagram and their corresponding relationships are used to construct the initial design of the system. The resultant initial design is shown in Figure 7. The entire initial design consists of 12 components with 56 operations. In the initial design of ehome, the component UserInterface handles the information displayed on user interface. The component UserManagement handles the user authentication details. The components DrapeDriver and DrapeRegulation control the movement of drapes. The components TemperatureRegulation and HeaterDriver handle the temperature control subsystem. The components MusicFiles, MusicSystem and SpeakerDriver enable the user to play music. The components CoffeeMachine and WaterControl control the coffee making activity.

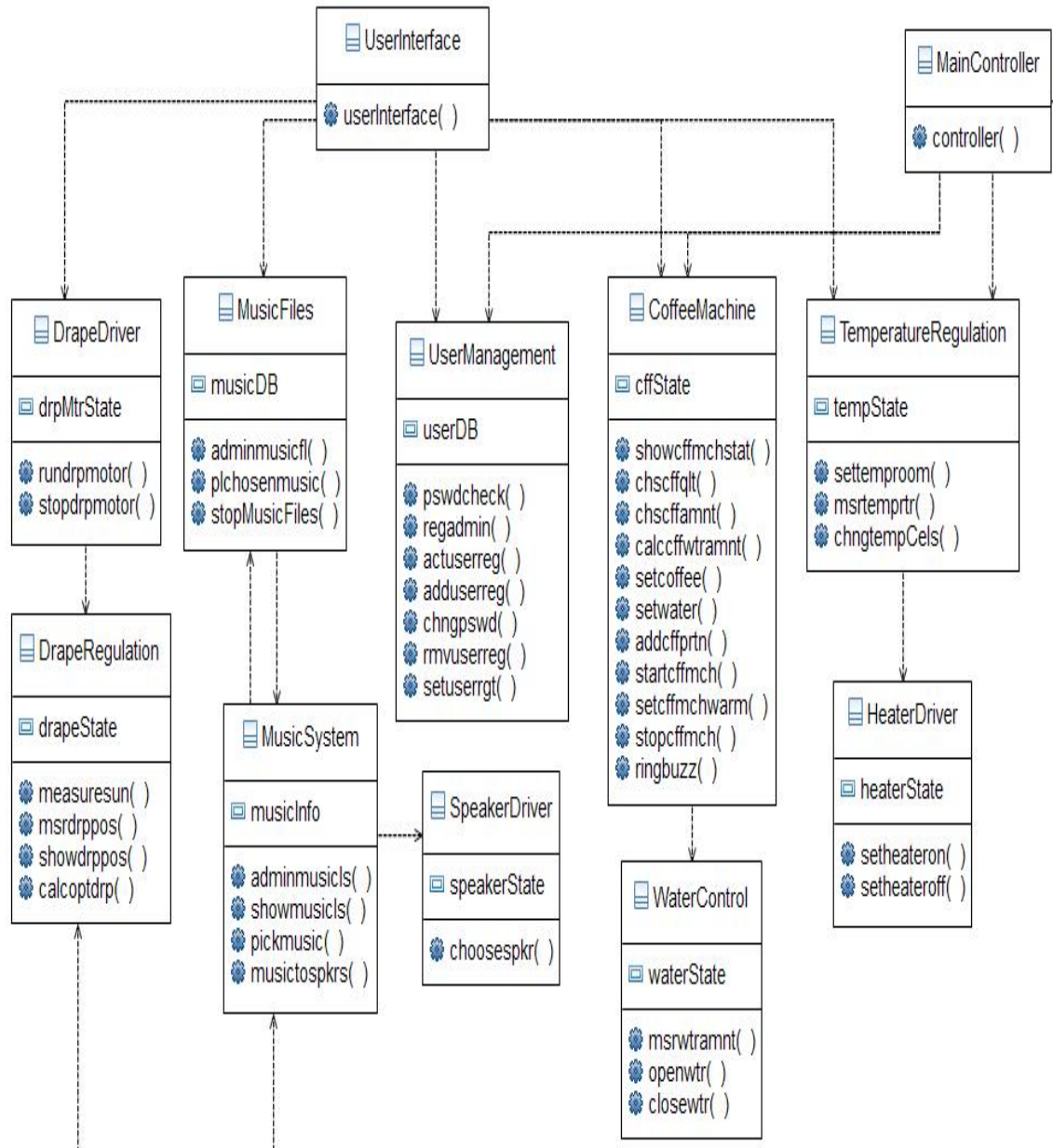


Figure 7. Initial design of ehome

Paper 1

H. Hadaytullah, S. Vathsavayi, O. Räihä, and K. Koskimies. Tool Support for Software Architecture Design with Genetic Algorithms. In *Proceedings of 5th International Conference on Software Engineering Advances (ICSEA' 10)*, 2010, IEEE Computer Society Press, pp. 359–366.

Paper 2

S. Vathsavayi, O. Räihä, and K. Koskimies. Using Quality Farms in Multi-Objective Genetic Software Architecture Synthesis. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC'12)*, 2012, IEEE Press, pp. 2130–2137.

Paper 3

S. Vathsavayi, H. Hadaytullah, and K. Koskimies. Interleaving human and search-based software architecture design. *Proceedings of the Estonian Academy of Sciences*, **62** (1), 2013, pp. 16-26.

Paper 4

S. Vathsavayi, O. Sievi-Korte, K. Koskimies, and K. Systä. Using Constraint Satisfaction and Optimization for Pattern-Based Software Design. In *Proceedings of the 23rd Australasian Software Engineering Conference (ASWEC'14)*, 2014, IEEE Computer Society Press, pp. 29-37.

Paper 5

S. Vathsavayi, O. Sievi-Korte, K. Koskimies, and K. Systä. Planning Global Software Development Projects Using Genetic Algorithms. In *Proceedings of 5th Symposium of Search Based Software Engineering (SSBSE'13)*, 2013, Springer LNCS **8084**, pp. 269–274.

Paper 6

S. Vathsavayi, O. Sievi-Korte and K. Systä. Tool Support for Planning Global Software Development Projects. In *Proceedings of IEEE International Conference on Computer and Information Technology (CIT'14)*, 2014, IEEE Computer Society Press, pp. 458-465.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3856-8
ISSN 1459-2045